CHALMERS
UNIVERSITY OF TECHNOLOGY

Development of an Anti-Pinch System for Passenger Vehicles

A Product Development Project at Volvo Cars

Master’s thesis in Product Development

MARCUS BOHLIN
GUNJAN NAGPAL

Department of Industrial and Materials Science

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2020
www.chalmers.se

MASTER'S THESIS 2020

Development of an Anti-Pinch System for Passenger Vehicles

An Anti-Pinch System that can replace multiple such systems used today, while also meeting legal safety regulations. In collaboration with Volvo Car Corporation.

Marcus Bohlin
Gunjan Nagpal

Department of Industrial and Materials Science
Division of Product Development
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Development of an Anti-Pinch System for Passenger Vehicles

An Anti-Pinch System that can replace multiple such systems used today, while also meeting legal safety regulations. In collaboration with Volvo Car Corporation.

MARCUS BOHLIN
GUNJAN NAGPAL

Supervisor: Egoi Sanchez Basualdo, Volvo Car Corporation, Interior Lights and Functional Parts

Examiner: Dr. Erik Hulthén, Chalmers University of Technology, Department of Industrial and Materials Science

Master’s Thesis 2020
Department of Industrial and Materials Science
Division of Product Development
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A lineup of Volvo Cars. © Volvo Car Corporation

Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Development of an Anti-Pinch System for Passenger Vehicles

An Anti-Pinch System that can replace multiple such systems used today, while also meeting legal safety regulations. In collaboration with Volvo Car Corporation.

A Product Development Project at Volvo Cars

MARCUS BOHLIN
GUNJAN NAGPAL
Department of Industrial and Materials Science
Chalmers University of Technology

Abstract

Products are becoming increasingly complex and so is every vehicle on the road. Passenger car manufacturers are adding more customer-oriented comfort features at every price segment in order to differentiate themselves from the competition. A lot of these features make use of automating the movement of various parts such as sunroof panels, seats and tailgates. Automation through motorized components needs to be accompanied by systems that can act as safeguards to prevent human injury due to this automation process. This thesis aims to lay the groundwork thereby equipping Volvo Cars with the knowledge to develop a consolidated anti-pinch system that could potentially be used in several different automotive components.

Different approaches for detecting pinch situations across industries and environments were evaluated. Two principally different approaches were chosen for further development, the first using hall sensors and the second leveraging ripple counting. A fundamental requirement for the developed concepts was conforming with legal regulations, namely the US regulation FMVSS No. 118 and the UN regulation ECE-R21-01. Taking an approach of early prototyping and testing, the concepts were continuously being tested along with their development.

After going through prototyping and testing, three different recommendations were made for Volvo Cars based on different timescales. A concept that meets legal testing requirements was suggested for implementation in 1-2 years, whereas other concepts that would require more developmental effort were suggested for timescales of 3-4 years and 7+ years. With an in-house system at their disposal, Volvo will have greater control over the functionality and safety of their anti-pinch systems, while also enabling cost savings.

Keywords: Automotive, Mechatronics, Passenger Safety, Anti-Pinch System, Hall Sensors, Sensorless Motor Control, Product Development

Acknowledgements

We would like to express our gratitude to our supervisor and examiner from Chalmers, Dr. Erik Hulthén, Associate Professor at Chalmers University of Technology, for his guidance throughout the project. We would also like to thank our supervisor from Volvo, Egoi Sanchez Basualdo, System Architect Interior Lights at Volvo Cars, for his support and constant motivation.

We are also grateful for the irreplaceable help and guidance received from David Johansson, Analysis Engineer at Volvo Cars, in the Strength and Endurance laboratory, and from Patrick Bobink, Hardware Designer DCU in Steering Wheel and Seat Electronics.

We would also like to thank our colleagues at Chalmers Technical University and Volvo Cars, especially interior light team and the HIL Laboratory.

Finally, we would like to thank all the people who were involved in the progress made during this thesis work for their contributions.

Marcus Bohlin and Gunjan Nagpal, Gothenburg, June 2020

Contents

1 Introduction

This chapter introduces the reader with the topic and the underlying motivation for conducting this thesis work. Some background information about Volvo Cars, the department where the thesis work was carried out and legal regulations is given. This is followed by sections explaining the aim, scope, and limitations of this thesis work.

1.1 Background

Volvo Cars is a leading personal passenger vehicle manufacturer based in Gothenburg, Sweden. The company is strongly associated with safety and innovation. It was founded as an affiliate of SKF (Svenska Kullagerfabriken), and the first mass-produced car “Jakob” was produced in 1927. Volvo Cars was a part of the Volvo Group until 1999 when it was sold to the Ford Motor Company, which remained the owner until 2010 when Geely Holding purchased Volvo Cars and became the current owner.

Today Volvo Cars has a presence in over 100 countries, has a workforce of approximately 38,000 employees, and in 2019 managed to produce over 700 000 vehicles in a year for the first time. The models in 2019 were – one hatchback, V40; two sedans, S60 and S90; two station wagons, V60 and V90; and three SUVs, XC40, XC60, and XC90. It should be noted that the V40 model went out of production in mid-2019. The headquarters and main R&D facility are located in Gothenburg, along with R&D offices in Shanghai and Chengdu in China. Volvo Cars is a part of the Volvo Car Group that also owns the consumer businesses Care by Volvo and M, Polestar, LYNC & CO, and Zenuity. (Volvo Cars Group, 2020)

The interior lighting and functional parts is a department within Volvo Cars that is responsible for the overhead console (OHC) including reading light lamps, general light lamps, mood light lamps, sunroof control, telematic buttons such as SOS/OnCall, and a host of indicators such as passenger airbag status, seat belt reminder, etc. The responsibility for the operations of the sunroof lies as the foundation for this master theses and is the main reason for that it is conducted within this department.

The underlying motivation for the topic of this master thesis is the unfortunate incidents of children dying or inflicting severe injuries in accidents involving motorized functions in vehicles, like with power windows or powered sunroofs. To address these safety issues, there are two legal motor vehicle regulations, the Federal Motor Vehicle Safety Standard No. 118 (FMVSS No. 118) in the United States and the United Nations Economic Commission for Europe ECE R21-01 standard in the rest of the world. This chapter will mainly address the FMVSS No. 118 regulation since it comprises of harsher requirements that also encompass the ECE R21-01 regulations.

According to the US Department of Transportation, National Highway Traffic Safety Administration (NHTSA) estimated that there were 6 deaths and 12 severe injuries related to power functions in vehicles per year in the USA before 2008. In addition to these, around 1900 less severe injuries like severed fingers, fractures, or bruises were treated by emergency rooms. There is also a large number of unrecorded cases of less severe injuries that did not get or require any medical treatment. Most of the deaths and severe injuries involve young children that were leaning on the door panel with their head or other parts of their body out through the window and accidentally activating the closing of the power window with a knee or foot and causing the window to close on the body part.

To prevent this type of accidents has a couple of legal actions been taken, first was the ball-test introduced which required that the closing of a power function could not be activated by a metal sphere with a radius of 20 mm ± 2mm that symbolizes a child’s knee (Cornell Law School, n.d.). After that was the requirement modified to only allow “pull-to-close” buttons, this means that the user is forced to pull the switch upwards in order to close the power window. These measurements were effective against accidents occurring when children were left alone in a vehicle but did not mitigate those that occurred when a person is operating the window without noticing that someone has a body part out through the window. This type of incident is referred to as “obstructed closing” by NHTSA and to deal with it were requirements on automatic reversal systems introduced which will overwrite the closing operation and reverse if a force of a certain magnitude is detected. The FMVSS No. 118 includes two main approaches to prevent these injuries, the requirements in the S4 paragraph are easier to fulfill but demands the attendance of the person opening or closing the window. The S4 paragraph must be fulfilled, otherwise the manufacturer is not allowed to sell the vehicle in the US. Meanwhile, the S5 paragraph is technically harder to satisfy and more costly for the manufacturer but allows for a much higher degree of design freedom like operating the functions remotely or via smartphone applications, etc. (National Highway Traffic Safety Administration, 2009)

1.2 Problem Description

At present, Volvo Cars sources the subsystems for power functions like the power windows, powered tailgate, electric sunroof, etc. from different suppliers. Along with the subsystem(s), the supplier also provides the safety system that prevents pinching accidents during the operation of the subsystem. For example, the supplier of the power window provides the pinch prevention system along with the components that are used to drive and control the operation of the window. In most instances, these components are placed together in a separate module for which the supplier has the design responsibility, and that module can easily be mounted on the Volvo assembly line. Due to this, several different “anti-pinch systems” from various suppliers are used in a vehicle, for example, one system for the power windows, another for the tailgate, and so on. Utilizing several different systems that mainly perform the same function can be an inefficient solution. As these suppliers provide such systems to several automobile manufacturers, the anti-pinch systems are generic and not specific to Volvo Cars. Sometimes resulting in non-optimal performance and an expensive and time-consuming calibration processes. These anti-pinch systems are very similar to each other, with only some modifications to fit the specific application. To this end, Volvo Cars ends up paying multiple times for the same or similar product. In addition to this, there are some performance problems, the most notable being that the electric sunroof system does not meet the legal requirements in the S5 paragraph in the US FMVSS No. 118, preventing Volvo from being able to use smartphone applications or rain sensors for controlling the sunroof in vehicles sold in the United States.

1.3 Aim

The aim of this thesis work was to develop an anti-pinch system that could work with all applications in a vehicle and that is compatible with the existing systems and standards in use by Volvo Cars. The developed system should meet the harsher US regulation FMVSS No. 118 to allow operations via the Volvo app. By developing this system, Volvo Cars aims to be able to reduce expenses since they would not need to buy the anti-pinch systems from multiple suppliers and avoid costs associated with their calibration. In addition to this, standardization could help in reducing the number and variation of parts that are needed to be stocked in the inventory. An in-house developed system would also provide Volvo Cars much more granular control over the functioning of the anti-pinch systems, creating room for improvement and thus, better safety and comfort for passengers.

1.4 Limitations

A couple of limitations are made in the project, mainly due to the project time of 20 weeks. The development work focuses on the software and no hardware components will be developed, instead will of the shelf hardware components be used in the project. The project is only focusing on the anti-pinch and the position control systems, no development effort will be put into reducing the risk of an accident by changing the button layout like pull to close.

Due to the limited time available to carry out this work, the system was only developed for the sunroof and the power windows, no further applications were investigated. The testing of the developed system was carried out only in lab conditions and was not implemented in a real vehicle.

No exact budget is set for the project and requests for funds were done throughout the project when needed.

1.5 Scope

The scope of the thesis was to develop a proof of concept for an anti-pinch and position control system that can be used in multiple areas within a vehicle and meet the legal requirements imposed on the system. A report that gives account for the development works should be delivered to Volvo Cars and Chalmers University of Technology. The project was carried out at Volvo Cars R&D center in Torslanda, Sweden, and ran for over 20 weeks during the two first quarters of 2020.

2 Methodology

In this chapter, the different stages of the project have been explained and the work has been carried out in these stages is also presented. In addition, the methods and tools that have been used in these phases have been shortly described.

The project was divided into seven major phases to make it easier to manage the project, with each phase having its own deliverables. Some of the phases were active concurrently to utilize time in the best possible manner. In Table 1, the phases are listed along with some of their most important deliverables.

Table 1 The seven major phases of the project and some of the most important deliverables for each phase

#	Phase	Major Deliverables
1	Planning	Planning report Initial time plan Risk severity matrix
2	Research and Technology Study	Technology Study Chapter
3	Customer Needs Study	Customer Needs List Customer Needs Metrics
4	Concept Development	Morphological Matrix Elimination Matrix
5	Prototyping	Prototypes for testing in different stages of the development process
6	Prototype Testing	Test report for the concept(s)
7	Concept Evaluation	Pugh Matrix Kesselring Matrix Final Concept(s)

2.1 Planning

The first phase of the project was dedicated to planning, starting with a planning report. The original time management plan was formatted into a Gantt Chart using Microsoft Project. One big part of the planning phase was the initial risk assessment. The purpose was to identify risks that could affect the project and come up with risk contingency plans where it is decided what should be done if a risk should occur. Through this planning, the impact of the events could be reduced. Risk assessment and management was also carried out continuously throughout the project. In addition to this, stakeholder management was also a part of the planning phase, when all the parties that had an interest in the outcome of the project were listed and assessed. Initial meetings were also held with the stakeholders to understand their expectations related to the project.

2.2 Research and Technology Study

The information gathered in the research and technology phase formed the foundational knowledge that all later development activities were built upon. This provided an understanding of how the market and products looked at the time. The phase mainly consisted of secondary research, which meant gathering information from academic papers written within the area and the legal regulations that affect anti-pinch systems. Information from patents and similar technologies within other areas where anti-pinch systems exist like elevators and machine tools was also included. For organizing the research, a literature data extraction worksheet was used, where all sources of information were listed along with their author(s), publication year, quality of content, relevance to the work, etc. To organize the patent landscaping was a patent data extraction sheet used. It includes the keywords, strings, patent classes and how many hits the searches gave. In this spreadsheet were also information from the most relevant of the identified patents stored. For the patent landscaping, the databases Espacenet and Google Patents were used.

2.3 Customer Needs Study

Identifying the customer needs and involving them in the development process is of utmost importance for a project to be successful. In this project were the customer needs identified by conducting semi-structured interviews. Mainly with Volvo personnel from the groups that have the technical responsibility for areas where an anti-pinch system is required and with the lab that perform the tests of the systems. An interview guide was prepared before the interviews to make sure that the most relevant topics are discussed. In addition to the interviews were semi-structured observations performed to get an understanding of how the current systems perform.

The customer needs study and the legal documents resulted in a customer needs list with wishes and demands. The customer needs list was used throughout the development process to make sure that the developed concepts aligned with the customer needs.

2.4 Concept Development

The development done in this project followed the process described in Product Design and Development by Ulrich and Eppinger and started by identifying the sub-problems that the system needs to solve. Then finding solutions to these sub-problems, the solutions were then organized in a morphological matrix where a great number of concepts could be generated. The generated concepts were then taken through an evaluation process where the first step was the Elimination matrix.

2.5 Prototyping

Prototyping was used as a tool throughout the entire project with quick and dirty prototyping in the early stages to understand technologies or specific functions. As many concepts as possible were built to enable performance testing and data gathering from multiple concepts.

In the prototyping was existing vehicle hardware used to the greatest extent possible in order to reduced cost and time consumption, this also included electronic components in the vehicles. Components that didn’t exist in the current vehicle was bought off the shelf. The focus of the prototyping was on the software and the interface between the software and the hardware, not on hardware development.

2.6 Prototype Testing

The developed prototypes went through a formal testing procedure performed by the Strength and Endurance Laboratory at Volvo. The testing procedure is standardized in the legal document ECE R21-01 and the FMVSS No. 118 and is the same for all anti-pinch systems. The results from these tests gave a fair and unbiased data that could be used to evaluate the performance of the developed concept(s) to the anti-pinch systems bought from suppliers and used in the current Volvo models. It also provided important information about the concept performance that was used in the concept screening and scoring instead of taking decisions based on estimates.

2.7 Concept Evaluation

Based on the research done and the results obtained during the prototype testing was the performance of the remaining concepts screened against a reference solution in a Pugh matrix, the customer needs will serve as criteria's in this screening. The Pugh matrix was followed by the Kesselring matrix, which is a scorning method where the customer needs are weighted according to their importance. Based on the results from the evaluation were few concepts recommended for further development.

3 Risk Management

This chapter gives account for the risk management activities that were undertaken during the course of the project. The importance of risk management in general is elaborated and after that are the identified risks, their consequences and what should be the response to these risks presented.

Risks exist in every project and poor risk management is often ranked as one of the most common reasons for why projects fail (Nelson, 2007). One important first step within risk management is to identify the areas where risks could occur and what the response should be if those risks occur. By doing this, there is a better chance to manage the situation if the risk occurs compared to if there were no guidelines for response. Some problems are unknown and caused by events that are hard or impossible to predict or prepare for beforehand, making such risks harder to handle. By being aware of risks in general and letting risk management be an ongoing process throughout the project, there is a possibility of a faster and more efficient response to these unknown risks as well. The process of identifying risks is an ongoing process that is active throughout the entire project.

There were four categories within which risks were identified. These categories were technical, project management, organizational and external. Technical risks are associated with technological aspects of the product like reliability and quality. Risks related to project management have more to do with time management and communication. Organizational risks consist of resources and their management. And external risks relate to factors like economy, changes in legislation, natural disasters, diseases, environmental hazards, etc. The identified risks along with their category, consequence, likelihood, impact, score, and response are listed in Table 2.

Some of the more important risks related to project management were about waiting times for parts, inputs, and organizational bureaucracy. These risks have a pretty high likelihood of occurring and their impact is significant as well. These risks can be mitigated by using methods such as concurrent engineering and by communicating timely. Organizational risks that were identified included risks such as no availability of an appropriate workplace for prototyping and testing. This could have happened during times when all facilities are fully booked due to ongoing projects and the impact could affect the time taken to finish the prototyping and testing. Early communication is a good way of mitigating this risk. The biggest technical risk identified was the concept not being able to meet the legal requirements. Since not meeting the legal requirements would make the concept useless, so this is a very high impact risk. The likelihood of this happening would be quite low, since legal requirements are very well defined and known. To mitigate this, knowledge about the legal requirements should be gathered early on and testing methods should be adopted that can check whether the concept meets these requirements. The most significant external risk identified was the breakout of a disease and the impact it could have on the project, which although had very little likelihood of happening, could potentially have a huge impact on the outcome of the project.

Table 2 Risk identification matrix, in this matrix are all the identified risks categorized and combined with their expected consequence, their likelihood of occurrence, the assumed impact it would have on the project, and what actions are taken to respond to the risk.

# ID	Category	Risk Condition	Consequence	Likelihood	Impact	Score	Risk Response
1	Project Management	Lose a lot of time waiting for input / parts / company bureaucracy	Lose a lot of valuable time on critical path. Iterating/Changes can take very long time	4	3	12	**Mitigate:** Concurrent engineering, working on multiple tasks to prevent sitting and waiting, timely communication
2	Project Management	Lack of knowledge in the team (software) / Not able to find the correct knowledge	Product will not have desired performance, unnecessary time spent on learning & troubleshooting	3	3	9	**Accept:** Not so much to do except trying to learn
3	Project Management	Time estimation – Never managed a project on fulltime for so long, hard to know how much that can be accomplished	Not able to deliver the product with desired quality at the desired time	4	2	8	**Accept:** Cannot do so much to prevent this except by being flexible and listen to advice from supervisors
4	Project Management	Long lead times when ordering from Volvo laboratories	Won't receive the desired components when they are needed	4	2	8	**Mitigate:** Order as early as possible, Concurrent engineering
5	Organizational	Lack of interest / input from stakeholders. Not enough information regarding existing solutions, user requirements or interfaces	Development process will be based on insufficient information and not all functions will be addressed	2	4	8	**Mitigate:** Clear and open communication, presenting the benefits that they can get from being involved in the project
6	Organizational	Senior management is resistant to take on the risk that is connected to providing safety functions with harsh legal requirements	No support for implementing the developed concept even if it meets/exceeds the performance requirements	4	2	8	**Accept:** Not so much we can do
7	Technical	Final concept doesn't meet legal requirements	The concept will be useless	2	5	10	**Mitigate:** Knowing the legal requirements and test the concepts to make sure that they meet them
8	Technical	Concept only works for some parts of vehicle	Will not meet the scope of the project	3	3	9	**Mitigate:** Testing the concept and doing development for the affected parts of the vehicle
9	Technical	Inferior in performance to current solution bought from supplier	The concept would need to be developed further or be rejected	3	1	3	**Mitigate:** Benchmarking to understand the performance of the competing solutions

10	Technical	Product cost is too high for mass manufacturing	Concept would be rejected even if it meets/overperforms the requirements	2	4	8	**Mitigate:** Considering cost early in the concept evaluation process
11	Technical	The developed system cannot be integrated in a good way with the existing components in the vehicle	The system would need modification or be rejected	2	4	8	**Mitigate:** Considering compatibility early in the concept evaluation process
12	External	Suppliers of current systems have a negative attitude to the project	Suppliers refuse to participate in the project	3	1	3	**Accept:** Nothing can be done
13	Organizational	No available space to do the practical development and testing of the project	The development process can become ineffective and testing insufficient	3	3	9	**Mitigate:** Early communication
14	Organizational/ External	Leak in the roof	Can destroy electrical equipment	1	3	6	**Mitigate:** Move the electrical equipment away from the sections in the roof that leaks.
15	External	Breakout of a pandemic of a new strain of a coronavirus	Could lead to the closure of the company or affect project members	1	5	5	**Accept:** Comply with government and healthcare regulations
16	External	Company shut down due to COVID-19 virus	Reduced time for development work, longer lead time and reduced access to Volvo resources	3	3	9	**Accept:** Comply with Volvo and healthcare regulations

4 Customer Needs Analysis

This chapter presents the structure, summaries and customer needs that were extracted from the interviews and observations. A detailed description of the FMVSS No. 118 is also given along with other legal requirements relevant for this project. In addition to this, the list of customer needs and the metrics against which they were evaluated have been detailed.

4.1 Interview Structure

The interviews conducted in this project aims at including the identified stakeholders in the development work. When gathering customer needs it is desired to interview the end-user or customer to understand their point of view. But for this project, the final product is a function in a vehicle that does not attract user attention, unless the function does not work, thereby causing injury. The amount of work needed to identify these cases, contact, and interview the affected users was considered to be excessive in relation to the input they could provide. Instead, the people within Volvo Cars that are responsible for the anti-pinch system in a specific area of the vehicle were treated as the internal customers and interviewed so that their knowledge could be utilized in the development process. Secondly, the interviews were conducted in the form of a normal office meeting, with an interview guide being used as the meeting agenda to allow for a semi-structured discussion. The interviews started with some background information about the project and its aim, and then following a funnel structure, general questions were asked about the previous and current anti-pinch systems that were used and the relationship with the suppliers. Then narrowing down to more specific questions about the technology and specific areas of improvement. A lot of emphasis was put on giving the interviewees the opportunity to speak freely and provide information even if no specific questions were asked within an area. Notes were taken during the interviews for documentation and more detailed summaries were written directly after.

4.2 Interview Summary

Anti-pinch systems do not have the same importance for all areas with moving parts. For the sunroof and power windows, the anti-pinch system is properly tested and there is a continuous process of improving both cost and performance of these systems. On the other hand, very little thought given to the anti-pinch system in other areas like the tailgate and the seats. The following sub-sections contain the summaries from all the conducted interviews and the customer needs that were derived from the interviews are listed, along with a short explanation of the need in Table 3.

4.2.1 Sunroof

This meeting was conducted at the department of interior lighting and functional parts, the same department that commissioned this project. The department is responsible for the operations of the electric sunroof and its anti-pinch system.

The main problem with the existing solution in the sunroof is directly connected to the extensive calibration process that is required for each model, model year and if one model is produced in multiple factories are calibrations done on a vehicle from each factory due to uncertain supplier and manufacturing variance. Even a change in the tooling used in the manufacturing or assembly process of the sunroof can require new calibrations. The calibration process is both time-consuming and expensive since it is done by the supplier at another location. The entire roof solution including the anti-pinch system used in current models are provided by one supplier, to protect intellectual property and “know-how” is Volvo’s insight in technical details limited. This lack of insight also makes it hard to derive expenses and identify what the exact price that Volvo pays for certain functions is, for example, the anti-pinch. The high price tag for such a simple system is motivated by that the supplier will take the legal responsibility if a pinching accident would occur.

When one supplier is responsible for such a large portion of the vehicle is it more difficult to replace them or negotiate the price. This is due to the cumbersome and expensive process of identifying a new supplier that can deliver the required quality and volume.

In addition to this are there some performance issues with the existing solution. Mainly, it does not meet the legal requirements in FMVSS No. 118 paragraph S5. The consequence of this is that the sunroof cannot be controlled using the Volvo mobile application in the US. The employees at Interior Light and Functional parts believe that this issue could be fixed relatively easy. For example, by reducing the speed of the panel in the areas where the requirements on the stopping distance are the harshest. Or by using ripple count instead of hall sensors to get a system that can react faster to a pinch situation. One other strong opinion is that the legal responsibility in case of an accident might be taken by the supplier, but it would still have a very negative impact on the brand Volvo. It is believed that it exists a general fear or unwillingness within the organization to take responsibility for functions that are governed by strict legal requirements and that this lies as a foundation for the unnecessary high price tag is accepted. It is also believed that this project can a step towards changing this mentality.

4.2.2 Power Windows

This meeting was conducted with personnel from the department of Switches and Door Electronics which among other things are responsible for the control of the power windows where the anti-pinch system is included. The supplier situation looks a bit different here compared to the sunroof, there is no system supplier that is responsible for an entire system. Instead is one supplier responsible for the glass, the window lift regulator is another and the door DCU is a third one. These suppliers tend to be the same for most or all models within one platform. This leads to that the suppliers have a much smaller area of responsibility and that Volvo gets better insight in what they are doing and can control the costs in a more efficient way. It is also easier to replace suppliers that cannot meet the requirements. It is required by all suppliers that the system should fulfill the legal requirements in FMVSS No. 118 paragraph S5. To guarantee that the process variation won’t make some cars exceeding this limit is Volvo requiring a maximum force of 75 N, which all suppliers currently meats. The systems are based on hall sensors and the current consumption for both position control and to detect pinches. It works fine for the windows mainly due to that they reduce the speed in the critical areas and that the inertia is much lower in the windows compared to the sunroof, so it is easier to stop is in a short distance. Using a system based on hall sensors is cost efficient and reliable, the only problem they experience at present is that the system detects false pinches. The cost for the system is not considered to be too high, even if cost reduction is a never-ending process is this not a focus area for the moment.

The calibration process has a very noticeable impact on the power windows as well, here is it the many variations that are the problem. At present there are two different window options per model, laminated and tempered glass, that all are different in size, thickness, weight and these should be combined with different types of seals where all combinations need to be calibrated. The calibration needs to be done for both the front and rear windows since they differ a lot from each other. In addition to this is smaller calibration fixes needed approximately once a year due to unknown manufacturing parameters that seldom can be identified.

Also here is the general opinion that Volvo could develop the anti-pinch system on their own and that it is the unwillingness within the organization to take the legal responsibility that is the reason for why they aren’t doing it.

4.2.3 Tailgate and Seats

The anti-pinch system in the tailgate falls under the responsibility of the art Security and Body Electronics, but the situation is very different here compared to in the power windows and the sunroof. There is an anti-pinch system and there is a requirement that the tailgate movements shouldn’t cause any injuries, except from this is everything handled by a supplier that provides the system for all models, no testing of the system is done by Volvo.

For the seats, there was no anti-pinch system in the current models. There are plans to include an anti-pinch system in models based on the SPA2 platform.

4.2.4 Test Lab

The strength and endurance laboratory at Volvo performs the testing of the anti-pinch system in the sunroof and the power windows. The testing is done both against the European ECE R21-01 and the US FMVSS No. 118 legal requirements for the windows, but only the ECE R21-01 for the sunroof. The tests are performed only in room temperature, with all other testing done by the supplier.

For some of the endurance and temperature tests that are done on the sunroof and the power windows, it is required to disable the anti-pinch system so that the testing can be performed correctly. Due to Volvo’s lack of low-level control over the anti-pinch system, this must be done by the supplier. This is time-consuming and frustrating for the personnel working in the lab and they wanted to be able to go in and disable the system themselves, something which they would be able to do if Volvo developed their own anti-pinch system.

Table 3 Customer needs identified during the interviews

Need No	Need	Explanation
2	Maximum pinch force is lower than the Volvo demand of 75 N	To guarantee that the process variation won’t make some cars exceeding the legal limit is Volvo requiring a maximum force of 75 N instead of 100 N
4	Detect pinch over the entire range of motion	To prevent the risk of a pinching accident, even if the pinch occurs outside of the range for the anti-pinch system as stated in FMVSS No. 118
8	The system cost per vehicle should be lower than for the current solution	Cost reduction is a continuous process and currently is the calibration standing for a big portion of the cost. Due to the mass manufacturing aspect can a small increase in cost for a component have large effects on the overall system cost.
9	The system should work in all-weather condition	Volvo Cars sell vehicle that can operate in all-weather condition therefore must the safety systems work in all-weather condition as well
10	The system should work in all climates	Volvo Cars sell vehicle that can operate in all climates therefore must the safety systems work in all climates as well
11	The system should be adaptable to wear and debris	After a number of opening and closing cycles will seals start to wear. This affect the friction against the moving object which in turn affects the working conditions for the motor. This should be considered by the system so that the threshold curve can be adjusted and keep the same level of sensitivity
12	The system should not detect false positive pinch situations	The anti-pinch system should not be activated by changes in the working conditions of the motor which are not caused by an object blocking the path of the moving object
13	It should be possible to disable the system when the vehicle is in factory mode	For some of the endurance and temperature tests that are done on the sunroof and the power windows is it required to disable the anti-pinch system so that the testing can be performed correctly
14	The system should be able to do a final calibration by itself for every individual vehicle when in factory mode	Due to the process variance will every vehicle be unique. The system should do a final calibration for the vehicle on its own
15	The system should meet Volvo’s endurance testing standards	The system should meet the endurance requirements set by Volvo Cars
16	The system should operate in real time	The system must process real time data continuously so that it understands the current situation
17	The real time system must have a deadline of 100 ms	To prevent lag between an input signal and the reaction from the system
18	The real time system should have a deadline of 10 ms	To prevent lag between an input signal and the reaction from the system
19	The system should be compatible with existing hardware in the vehicle	The developed system should not require any hardware changes in the vehicle to avoid extra cost
20	The system should be compatible with existing software in the vehicle	The developed system should be compatible with the software system used to control the specific part of the vehicle. Example: LIN
21	The system should detect an object before colliding with it	To prevent causing any harm to the object and to avoid false positive pinch detections should the system have a direct system that can detect objects before they come in mechanical contact with the moving object
22	The system should not increase the noise level in the vehicle	To preserve a good noise level for the user, the system should not increase the noise level

4.3 Observation Structure

To get an understanding of how the current anti-pinch systems worked in a Volvo vehicle, an observation study was performed. The observation was structured to be simple and took approximately one hour to complete. The examined vehicle was a Volvo V90 Cross Country. The areas that were observed were the sunroof, front and rear windows, the seats and the tailgate.

4.4 Observation Summary

In the sunroof, it was observed that the anti-pinch system worked moderately well. The anti-pinch systems in both the panel and the curtain relied on indirect methods to detect a pinch situation. The force required to trigger the anti-pinch system was perceived as higher for the curtain than the panel. The pinch force seemed to vary over the range of motion with lower forces when the panel or curtain was closer to the closing edge. When the anti-pinch system got activated, the panel retracted by a few centimeters not to the fully open position. The overall impression was that the system works but could be more sensitive.

The anti-pinch system in the windows worked the best. The force required to initiate a pinch differed depending on the position of the window. The system was more sensitive when the window was close to its fully closed position. When the anti-pinch system got activated, the window rolls down completely, instead of just a few centimeters like in the sunroof. The overall impression was that this system worked very well. The system was very sensitive in the critical zone close to the upper edge.

The anti-pinch systems in the tailgate worked moderately well. Contrary to expectations, no direct sensing was present, and the system depended only on indirect sensing. The force required to trigger the anti-pinch was quite high. When the anti-pinch system was triggered, the tailgate stopped right at the position where it was triggered and a beep from the center console was sounded. The overall impression was that this system required the least possible sensitivity to get approved.

Despite the seats having motorized functions inside of the vehicle which could cause pinching accidents, no anti-pinch system was observed. According to Volvo, an anti-pinch system was not used in the seats on the current SPA1 platform, but there would be some anti-pinch functionalities for the seats in the next generation platform.

4.5 Legal Requirements

A majority of the most important demands for this project come directly from the governing legal document FMVSS No. 118. As described in the This chapter introduces the reader with the topic and the underlying motivation for conducting this thesis work. Some background information about Volvo Cars, the department where the thesis work was carried out and legal regulations is given. This is followed by sections explaining the aim, scope, and limitations of this thesis work.

Background chapter, there is also the United Nation’s legislation ECE R21-01 which is very similar to FMVSS No. 118 but is easier to fulfill. In the following section, the relevant paragraphs of the FMVSS No. 118 have been explained and the customer needs derived from the sections have also been listed.

The FMVSS No. 118 regulation consists of nine paragraphs, out of which paragraph S4, S5 and S8 affect this project to a great extent. In short, the paragraphs are as follows; S1. *Purpose and scope;* S.2 *Application;* S3. *Definitions;* S4. *Operating requirements;* S5. *Automatic reversal systems;* S6. *Actuation Device;* S7. *Test procedure;* S8. *Test rods;* and S9. *Procedure for measuring infrared reflectance of test rod surface material (*(National Highway Traffic Safety Administration, 2011). The complete legal document with all nine paragraphs can be seen in Appendix A.

Paragraph S4 is about the basic level of protection. It states that functions relying on power supply from the vehicle, like windows and roof panels, can only be activated if the key is in the ignition and in the "On", "Start" or "Accessory" position. Also are listed, some other cases where a remote control can be used from a limited distance range, but these exceptions still require the operator to be in close proximity to the vehicle. The purpose of this paragraph is to ensure that children cannot operate the power functions without the presence of an adult (National Highway Traffic Safety Administration, 2009).

Paragraph S5 allows for a different approach to minimize the risk for accidents related to power functions by introducing an "automatic reversal system" (ARS). If the ARS system meets the requirements set in S5, it no longer needs to comply with the restrictions stated in paragraph S4. Which means that the power functions can be operated remotely, without the presence of the ignition key. The only demand that is stated in paragraph S5 is that if a pinch is detected, the window or panel must stop and reverse for a minimum of 0.015 seconds before the pinch force exceeds 100 N. In addition, the paragraph is connected to test and test rod conditions explained in paragraph S7 and S8, which makes the pinch force demand hard to fulfill. The biggest difference compared to the ECE R21-01 is that the test rod used to measure the pinch force should be placed angled relative to the window or panel to make it more difficult for the system to detect the test rod as seen in the middle example in Figure 1, in ECE R21-01 are the test rod always placed perpendicular to the moving object. (National Highway Traffic Safety Administration, 2009)

Paragraph S8 specifies details about the test rods that should be used when performing a test on an ARS system, making it an important factor for deciding customer needs. A test rod must be between 4 and 200 mm, indicating that the ARS system can be deactivated outside of these limits. Each test rod must have a force-deflection ratio of 65 N/mm if the diameter is less or equal to 25 mm; and 20 N/mm if the diameter is greater than 25 mm. This means in practice that for rods of diameter ≤ 25 mm, a pinch force of 100 N will be obtained when the window or panel moves $\frac{100\,N}{65\,\frac{N}{mm}} = 1.54mm$ and a pinch force of 75 N, which is the maximum that Volvo allows, will be obtained after 1.15 mm. For test rods > 25 mm, these distances are 5 mm and 3.74 mm, respectively. The placement of test rods for three different applications can be seen in Figure 1 and customer needs derived from the legal documents can be seen in Table 4.

Table 4 Needs extracted from the legal documents

Need No	Need	Explanation
1	Maximum pinch force is lower than 100 N	Specified in FMVSS No. 118 paragraph S5
3	Detect pinches in the range between 4 and 200 mm from the frame	Specified in FMVSS No. 118 paragraph S8
5	Reverse direction on pinch	Specified in FMVSS No. 118 paragraph S5
6	Meet force requirements using a spring stiffness of 65 N/mm for the last 25 mm	Specified in FMVSS No. 118 paragraph S8
7	Meet force requirements using the spring stiffness of 20 N/mm for distances greater than 25 mm from the frame	Specified in FMVSS No. 118 paragraph S8

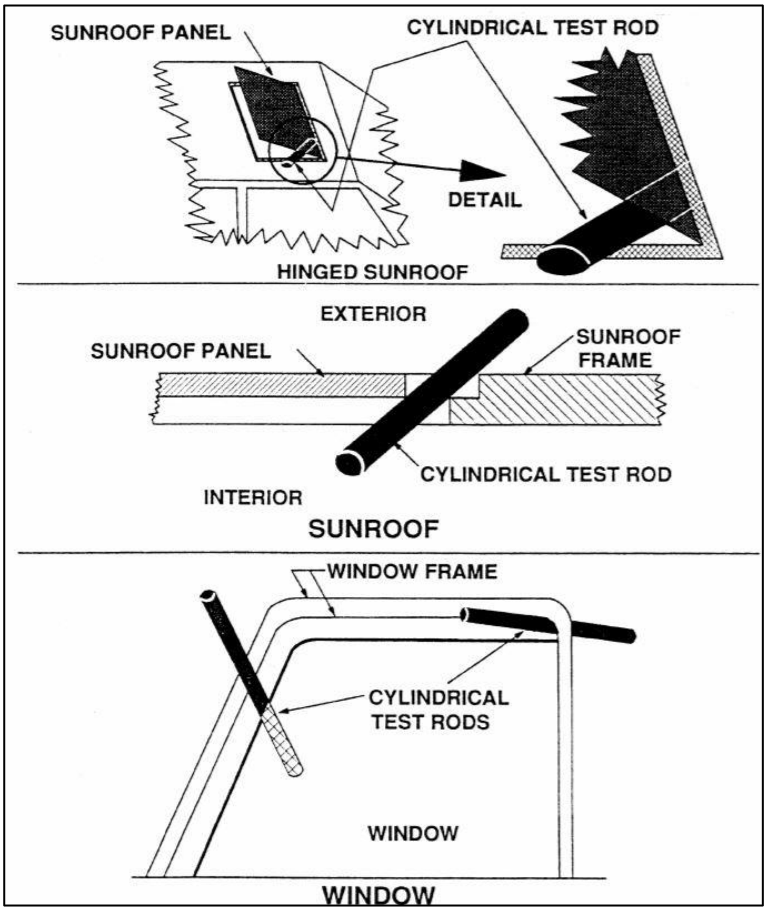


Figure 1 Test rods placed in different positions for pinch tests of sunroofs and power windows (National Highway Traffic Safety Administration, 2011)

4.6 Customer Needs List

The interview and observation phase ends with defining a list with customer needs that are important for the success of the product. These customer needs are often expressed in general terms when it comes to language, often called “language of the customer”. In this case, when the interviewees are relatively familiar with the subject and the technology, the needs become more concrete. In the customer needs list, the needs have been divided into two different classes: “D” demands that must be fulfilled; and “W” wishes that should be fulfilled as good as possible. The wishes are in turn ranked on a scale from 1 to 5, where 1 is the least important and 5 is the most important. A wish with a rank of 5 is as close to being a demand as possible.

What rank a wish should get was determined based on the information gathered in the interviews and the observation. In addition to this, the needs have been divided into categories depending on which area that the need was relevant for, these categories were: *Legal*, referring to needs derived directly from the FMVSS No. 118 and ECE R21-01; *Standards*, for needs connected to Volvo or universal standards; *Operations*, for needs related to the operations and performance of the system; *Economics*, for cost-related needs; and finally, *Manufacturing*, for needs related to the manufacturing and development process at Volvo. The complete customer needs list can be seen in Table 5.

Table 5 Customer needs

No	Need	W/D	Imp	Category
1	Maximum pinch force is lower than 100 N	D	D	Legal
2	Maximum pinch force is lower than the Volvo demand of 75 N	W	4	Standards
3	Detect pinches in the range between 4 and 200 mm from the frame	D	D	Legal
4	Detect pinch over the entire range of motion	W	4	Operation
5	Reverse direction on pinch	D	D	Legal
6	Meet force requirements using a spring stiffness of 65 N/mm for the last 25 mm	W	5	Legal
7	Meet force requirements using the spring stiffness of 20 N/mm for distances greater than 25 mm from the frame	W	5	Legal
8	The system cost per vehicle should be lower than for the current solution	W	5	Economic
9	The system should work in all-weather condition	D	D	Operation
10	The system should work in all climates	D	D	Standards
11	The system should be adaptable to wear and debris	W	4	Standards
12	The system should not detect false positive pinch situations	W	5	Operation
13	It should be possible to disable the system when the vehicle is in factory mode	W	3	Manufacturing
14	The system should be able to do a final calibration by itself for every individual vehicle when in factory mode	W	3	Manufacturing
15	The system should meet Volvo's endurance testing standards	D	D	Standards
16	The system should operate in real-time	D	D	Operation
17	The real-time system must have a deadline of 100 ms	D	D	Operation
18	The real-time system should have a deadline of 10 ms	W	3	Operation
19	The system should be compatible with existing hardware in the vehicle	D	D	Compatibility
20	The system should be compatible with existing software in the vehicle	D	D	Compatibility
21	The system should detect an object before colliding with it	W	3	Operation
22	The system should not increase the noise level in the vehicle	W	3	Operation

4.7 Metrics for User Requirements

The customer needs that have been listed in the requirements specification are general. To be able to quantify them and to make sure that the developed concept meets the performance targets, metrics and target values for the individual needs were set. The list with these metrics and target values can be seen in Table 6.

Table 6 Metrics and target values for the individual customer needs

Metric #	Need #	Metric	Unit	Target Value
1	1,2	Maximum detected pinch force	N	<75
2	3,4	A pinch is detected between 4 mm and 200 mm from the edge or over the entire range of motion	Binary	Yes
3	5	The moving object reverses after a pinch	Binary	Yes
4	6	Stopping distance for moving object using a spring with the spring force of 65 N/mm, for a distance ≥ 4 mm & ≤ 25 mm from the edge	mm	1.15
5	7	Stopping distance for moving object using a spring with the spring force of 20 N/mm, for a distance ≤ 25 mm from the edge	mm	3
6	8	Estimation of cost per vehicle for concept in mass production	SEK	151
7	9	Dust/water resistance	IP class	64/65
8	10	Operating temperature	°C	-40 to + 80
9	11	Meets pinch force requirements after Volvo lifecycle test	Binary	Yes
10	12	Number of false pinch detections in demanding operation conditions	No	0
11	13	User test, is it possible to disable system	Binary	Yes
12	14	System test if it can calibrate itself	Binary	Yes
13	15	Volvo lifecycle test	Cycles	Volvo Standard
14	16	System test	Binary	Yes
15	17,18	System time test	ms	10
16	19,20	System compatibility test	Binary	Yes
17	21	Pinch detection test, is contact needed for pinch detection	Binary	No
18	22	Sound level increase compared to old system	dB	0

5 Technology Study

This chapter presents the technologies relevant to this thesis work, starting with the motors and electrical drive. A lot of information is given about the sensors that could be used for detecting a pinch situation, both with and without mechanical contact. Additionally, technologies for signal filtering and communication protocols have been presented.

5.1 Driving Systems

The need for all anti-pinch systems arises from the risk associated with a moving object. To move an object, some sort of motor is required. For most applications that require a motor in automobiles like fans, pumps, etc., DC motors are used. This is also the case for the applications that are relevant to this project. Two different types of DC motors can be used, brushed (BDC) and brushless motors (BLDC). Diagrams of the two motor types are shown in Figure 2. The working principles for both types of motors are similar, there is one stationary part, the stator; and one rotating part, the rotor. The rotor is most often placed inside of the stator, but it could also be done in the opposite way with the rotor on the outside. These motors are called radial flux motors since the magnetic field travels in the radial direction through the airgap between the stator and the rotor (Microchip Technology Inc., 2004). In a brushed DC motor, the electrical windings are placed in the rotor. These windings are energized so that they create a magnetic field that can be attracted by opposite poles in the stator where permanent magnets or electromagnets are placed. As the rotor rotates, the windings are energized in different patterns so that the magnetic poles shift, and the motor keeps rotating. This is done mechanically by the brushes and the commutator; the commutator is a ring on the rotating axis where different sections are connected to the windings in the rotor. The brushes slide over the commutator and changes the direction of the current in the windings. Due to this sliding contact between the commutator and the brushes, there is a lot of wear on these parts which requires maintenance. These brushed DC motors are generally cheap, easy to control, and a well-tested technology that are used in a wide range of applications from toys to automotive (Hanselman, 2006).

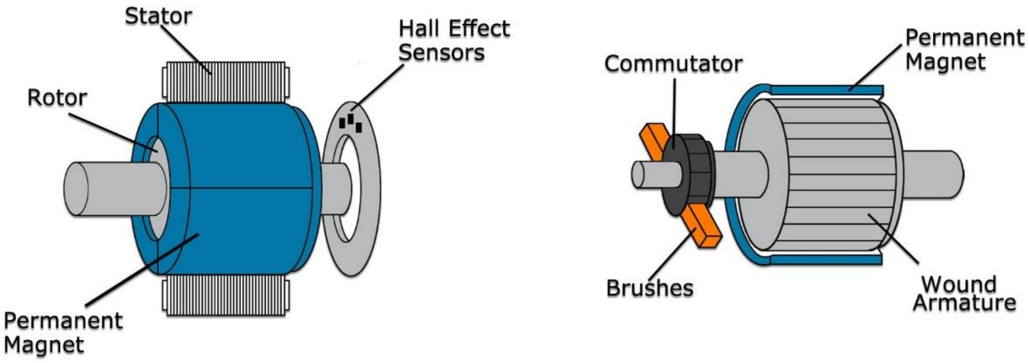


Figure 2: To the left in the figure can a brushless DC motor (BLDC) be seen and to the right a Brushed DC motor (BDC)

In brushless DC motors (BLDC), on the other hand, the windings are always placed in the stator and a controller is used to switch the polarity. This is a more complex and expensive system but has a lot of advantages since there is no mechanical contact needed, which eliminates almost all wear and maintenance. It is also much easier to cool the windings when they are on the stator (ATMEL Corporation, 2016).

5.2 Electrical Drive

Motors in automobiles are primarily driven by two different electrical driving systems. The first system is based on relays and the other system uses MOSFETs.

5.2.1 Relays

A relay can be termed as an electronically actuated switch. It consists of an electromagnetic coil, which when energized causes the switching in an external circuit. This allows isolation between a low-power circuit, for example with a microcontroller, and a high-power circuit, for example, a DC motor. Relays can be used in an H-Bridge configuration to drive a brushed DC motor in two directions, as shown in Figure 3.

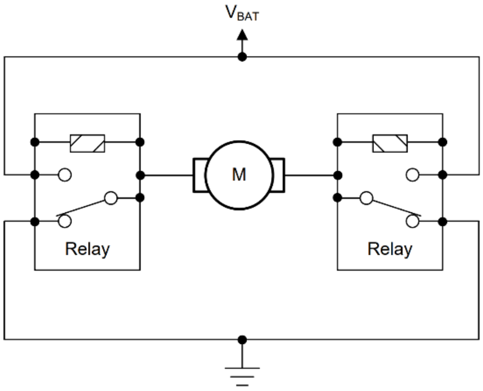


Figure 3 An H-Bridge arrangement using two relays for bi-directional motor control (Amin & Beard, 2016)

5.2.2 MOSFET Drivers

In contrast to relays, which are electromechanical devices, Metal Oxide Semiconductor Field Effect Transistors (MOSFETs) are electronic devices. They can act as switches that turn on or off based on Pulse Width Modulation (PWM) signals given to a gate driver. Similar to relays, it is possible to get bi-directional control of a brushed DC motor by using MOSFETs in an H-Bridge configuration, as shown in Figure 4. A MOSFET's rate of switching on or off determines how much current can pass through it. By controlling the current that is allowed to pass, speed control of a brushed DC motor can be achieved.

PWM signals, which instruct the motor driver for switching the MOSFET on or off, are generated by a microcontroller unit (MCU). A PWM signal is an on-off signal pattern, where the time duration for which a signal is ON is referred to as the width of the pulse, and the proportion of the ON time to the time period of the signal is called the duty cycle (expressed in %). PWM signals at different duty cycles from an Arduino microcontroller board are shown in Figure 5.

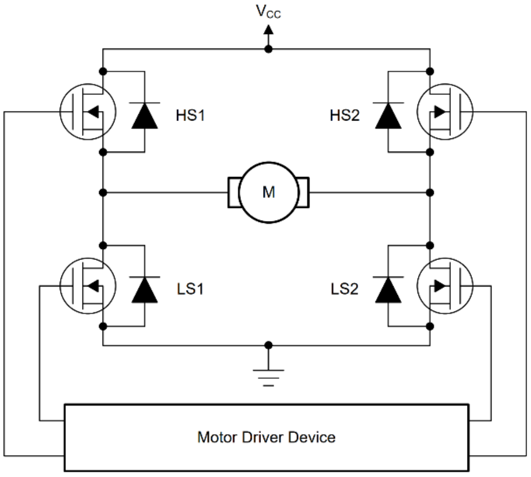


Figure 4 An H-Bridge arrangement using four MOSFETs for bi-directional motor control (Amin & Beard, 2016)

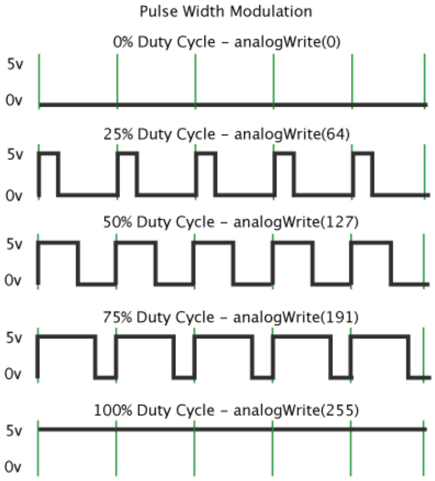


Figure 5 PWM signals from an Arduino at different duty cycles (Hirzel, n.d.)

5.3 Sensors for Detecting Pinch Situations

The primary methods for anti-pinch systems can be divided into two major categories, those systems that can detect an obstacle before any mechanical contact has occurred and those that require mechanical contact to be able to detect an obstacle. The following subsections are aimed at explaining the different sensors that can be used for detecting pinch situations.

5.3.1 Sensors not Requiring Mechanical Contact

These sensors can detect a pinch situation before it occurs without inflicting any force, making them very suitable for use in anti-pinch systems. They are also generally insensitive to vibrations, temperature variations, deformations, and wear, which has a significant impact on the reliability of the system over its lifetime. These sensors need to be mounted in such a way that they can monitor the area where a pinch could potentially occur, resulting in that the sensors often are separated from the module containing the motor, driving electronics, etc. This can lead to complications, as the module often is provided by a supplier which is responsible for that particular module but lacks the authority to change anything outside of it. In addition to this, extra wiring is required for the sensors to communicate with the control module. The most common types of these sensors are explained in more detail below.

5.3.1.1 Ultrasonic

These sensors use ultrasonic sound waves to detect the presence of objects in their vicinity. Sound waves are termed as ultrasonic when they have a frequency greater than 20 kHz, which is above the range of ‘sonic’ frequencies audible to humans. Ultrasonic sensors use the time-of-flight technique to determine the distance of a target object. A transmitting ultrasonic transducer sends a burst of ultrasonic sound waves, which get reflected from the target object’s surface. These reflected waves are called the echo. The echo is then picked up by a receiving ultrasonic transducer. The difference between the time of sending the waves and the time when the echo was received is measured. Since the speed of sound waves in the traveling medium is known, the distance of the target object can be calculated.

Since the distance of the target surface can be determined in a short interval of time, it can also be used as an input for pinch detection. A microcontroller can be used to track the distance of the target. When the user initiates the movement, the distance values must increment gradually in an expected behavior, which can be continuously verified by the microcontroller. If an abnormal change in the distance is detected, it would mean that there is an unexpected object in the path of the moving object which could get pinched. When this occurs, the microcontroller can stop the movement, preventing the pinch situation. A diagram illustrating the principle of distance measurements for ultrasonic sensors is shown in Figure 6.

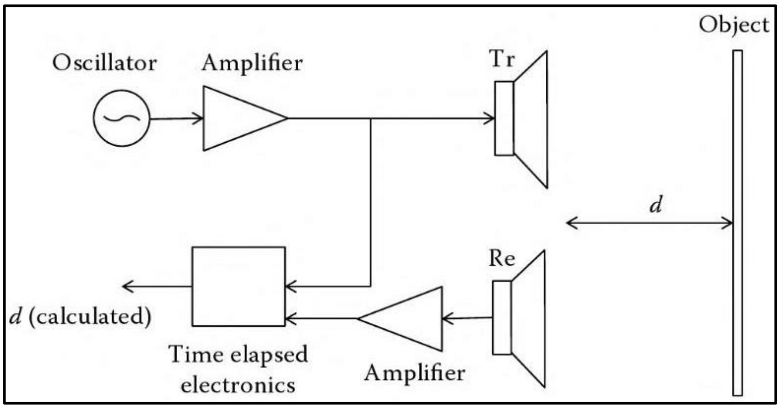


Figure 6 Principle of an ultrasonic distance measurement system (Webster & Eren, 2017)

Ultrasonic transducers needed for range finding applications consume low power and are very inexpensive but integrating them into highly space-constrained applications can be difficult. These sensors can detect a pinch before there is mechanical contact, but they have a minimum limit on sensing range under which they do not perform reliably (Nitsche & Herrmann, 2009). They can also be used as an indirect method, for example monitoring the speed of a window. More about indirect methods will be explained in later subsections.

5.3.1.2 Infrared

Infrared refers to the light signals that have a wavelength longer than the red light in the visible spectrum, generally between the range of 700 nm - 1 mm. Infrared sensors are used for night vision, wireless communication, astronomy, obstacle detection, etc. Infrared sensors are divided into two main categories, active and passive. Active sensors consist of a diode emitting light and a receiver. Infrared light is transmitted from the diode, reflected against the target object and then received by the receiver. By continuously repeating this process, the distance of the target object can be determined. Meanwhile, a passive sensor only has a receiver and is more suited for motion detection than for distance tracking. (Jost, n.d.)

5.3.1.3 LiDAR

LiDAR stands for Light Detection and Ranging. Like ultrasonic sensors, LiDAR makes use of time-of-flight calculations to determine the distance to a target object. However, instead of using ultrasonic sound waves, LiDAR makes uses of light beams, mostly infrared lasers. As a result, by using a signal of a higher frequency (430 THz - 750 THz), LiDAR acquires more accurate and reliable data than ultrasonic sensors (40 kHz – 10 MHz). The data from a LiDAR can be used to create a 3D image of its surroundings (Allodi et al., 2016).

A short-range LiDAR could be used as a system for interior monitoring. Intelligent algorithms have made it possible for computers to identify objects and humans using LiDAR data. These techniques could be used to develop an intelligent system that can identify passengers, windows, sunroof panels, seatbelts, etc., and determine risky behavior (Simons-Morton et al., 2005). Such a system could also be able to identify pinch situations. For example, if a passenger has their hands outside the window, and if the window close button is pressed, the system could be trained to recognize this as a pinch situation.

5.3.1.4 Capacitive

Capacitance is referred to as the ability of a system to store electrical energy or charge. The capacitance of a capacitor depends on the distance between the plates of the capacitor *(d)*, area of the plates *(A)* and the dielectric constant *(ε_r)*, and is defined as:

$$C = \varepsilon_r \frac{\varepsilon_0 A}{d}$$

where, ε_0 is the dielectric constant of vacuum, which is equal to 8.854188 x 10^{-12} F/m. Capacitive sensors generate electrical signals in response to changes in either of these three parameters. (Webster & Eren, 2017)

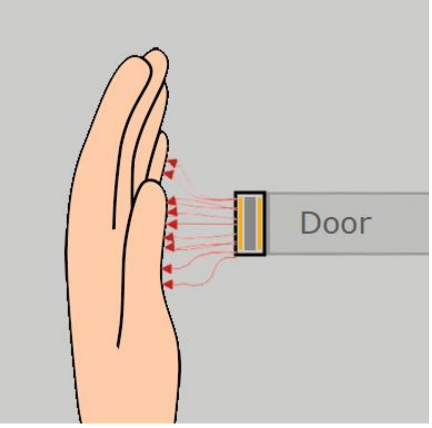


Figure 7 Changes in the electrical field of the capacitive sensor can detect presence of a human body part

Capacitive sensors can be used to detect the presence of objects without requiring mechanical contact, as shown in Figure 7. Capacitance-based proximity sensors rely on the changes to the frequency of an LC tank circuit, which is connected to a metal plate. A change in the oscillator frequency occurs when the target object moves within the capacitor’s electrical field. The changed frequency can be detected by a microcontroller unit and mapped to the relative or absolute position. (Terzic et al., 2012)

5.3.2 Sensors Requiring Mechanical Contact

As the name suggests, sensors in this category require that the moving object, a sunroof or window glass, actually comes in contact with the object that blocks its path, which can inflict some damage. The sensors used can be divided into two categories; direct systems, where a force sensor is located on the surface that will come in contact, and indirect systems, where the performance of the motor is monitored to detect abnormal behavior in various motor parameters, for example, the angular velocity, torque or back-emf. Indirect systems are mainly used in automotive applications since these solutions, in general, are the most cost-optimized alternatives. (ATMEL Corporation, 2016)

5.3.2.1 Direct Sensing

Pinch detection can be done through direct sensing of the physical movement of the object under the pinch forces. The movement can be detected by placing two parallel conductive surfaces in, for example, the door arc seal, which would come in contact with each other when an object presses against the seal. Connecting wires to these conductive surfaces would result in the joining of a circuit that is connected to a microcontroller, which would detect the signal as a pinch situation. Figure 8 shows three sketches of how a sensor in a seal might look.

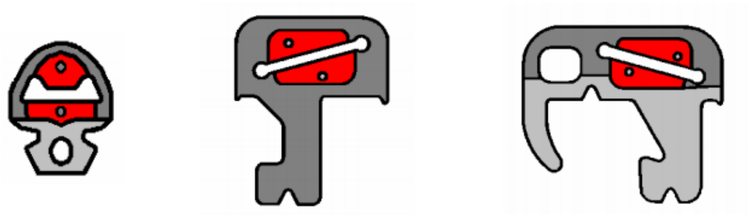


Figure 8 Different seal profiles with sensor (in red color) inserted (Sollmann et al., 2004)

Such a system would require minimal calibration and have less dependence on external factors, like the speed of the vehicle, the roughness of the road, wear and tear, etc., as only the physical interaction with the object would trigger the pinch situation. (Sollmann et al., 2004)

5.3.2.2 Indirect Sensing

Indirect measures are used by monitoring the motors that drive the system. This is the standard in the automotive industry since it is cost-optimized and gives a lot of design freedom as the systems can be placed elsewhere than in the contact region (Pak et al., 2017). The two leading technologies for indirect monitoring are Hall sensors and ripple counting, both of which are explained in the following sections.

5.3.2.2.1 Hall Sensor

One of the most common methods for indirect motor position control in automotive is to use a sensor that is measuring the Hall effect (Westgate & Chien, 1980). By monitoring the changes in the density of the magnetic field can the sensor determine when a motor pole is passing, this information is then converted to an electric pulse that is processed by a control unit. By measuring the width of and interval between the pulses, the rotational velocity of the motor or shaft can be calculated. Since the number of poles in the motor is known and how many pulses that are received per revolution can the hall sensors be used for position control as well.

There are mainly four types of hall sensors: latching switches, bipolar switches, omnipolar switches and unipolar switches. A latching switch is often used in an anti-pinch system due to its characteristics in controlling the switching between negative and positive poles. The sensor requires a positive and a negative pole to be able to operate. When the magnetic flux density from a positive pole (south) reaches a pre-determined value, the magnetic operate point (B_{OP}) of the device be gets switched on and stays in that state, even if the magnetic flux density decreases. It will stay switched on until a negative magnetic flux density from a negative pole (north) of adequate strength is detected, triggering the magnetic release point (B_{RP}). Then the sensor will switch to its turned-off state and stay there until the next B_{OP} value is detected. The B_{OP} and the B_{RP} are symmetric around $B = 0$, the neutral magnetic flux density field, but with opposing polarities. When used to monitor the movement of a motor, a latching switch hall sensor provides a pulse-train of ON and OFF pulses, which provides an effective way of controlling the motor. (Allegro MicroSystems, 2013a)

Bipolar switches were developed as a low-cost alternative to latching switches where the B_{OP} and the B_{RP} are not symmetric around B. It is now mainly used in applications that require high sensitivity or that both the B_{OP} and the B_{RP} are in the same polarity (Allegro MicroSystems, 2013b). The placement of a hall sensor around a motor shaft can be seen in Figure 9.

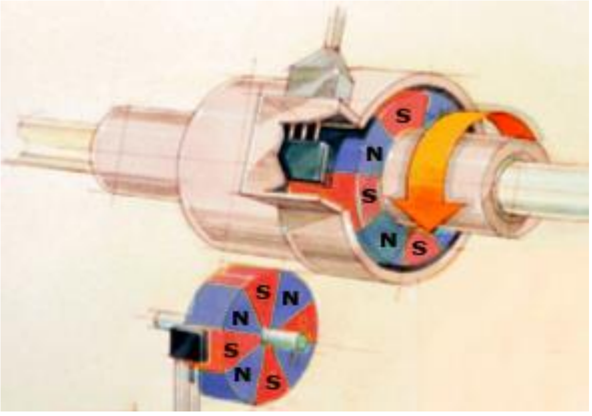


Figure 9 The placement of a bipolar hall sensor around the poles and the motor shaft (Allegro, 2013)

Unipolar and omnipolar switches are only operated by one polarity and are more suitable for translating applications rather than rotating. Hence, these cannot be used for motor control (Storr, n.d.).

Detection of a pinch situation starts by using the values received from the Hall sensor(s) and computing a physical quantity, like angular velocity, torque, torque rate, etc. Then this computed value is compared with reference values for that specific application. For example, if the angular velocity is lower than a threshold value determined based on a reference dataset, a pinch situation can be declared. The nature of these values depends on the motor parameters, friction between the moving parts and the seals, mass of the moving part, etc. Deciding which physical quantity to use is a tradeoff between required computational power and insensitivity towards measurement noise. Calculating the angular velocity requires the least computational power but is more sensitive to parameters that can vary over time due to changes in the environment and wear. Torque and torque rate are comparatively less sensitive to changes over a long time, but for their computation, a current sensor is needed.

The main advantage with Hall sensors is their cost-effectiveness, and their low maintenance as they are contact-free and are not affected by moisture or vibrations. The accuracy of the sensor depends on the number of poles in the motor and the number of sensors, since a minimum of two poles must pass the sensor to identify any change in velocity or pulse width. This can lead to undesirable lag before the sensor is able to identify a pinch situation (Rajaram & Murugesan, 1978).

5.3.2.2.2 Ripple Count

Measuring the motor speed using Hall effect sensors requires placing physical sensors along with the motor and separate wires for signal transmission. Thus, there is a need for sensorless methods of measuring motor speed and position. One way of achieving this is by measuring back electromotive force (back EMF) pulses that are generated in the motor (Sullivan, 2017). The back EMF is generated due to the motion of the motor windings inside the magnetic field generated by the permanent magnets. This back EMF produces a sinusoidal signal, the frequency of which is proportional to the speed of the motor. Another way of sensorless measurements makes use of the commutation and the construction of the motor. When the motor windings get in contact with the commutator, they effectively short and produce a ripple of current, thus making the frequency of this ripple also proportional to the speed of the motor (Consoli et al., 2004).

The motor's in-line current consists of DC and AC components. The DC component is the input current that drives the motor, and the AC component is made up of the back EMF and the commutation ripples. Also, there are some variations that can be observed in the current measurements. These can be attributed to many factors like changes in friction, wear and tear, temperature, etc. As a result, there is a need for filtering and conditioning the signal to make it useful for establishing accurate speed and position (Testa et al., 2014). This signal can then be fed into a microcontroller, which can compare the instantaneous speed and position data with a reference dataset. A threshold value can be associated with a pinching event and when this value is detected, the microcontroller can stop or reverse the motor.

Various studies have suggested different methods of filtering and conditioning the current signals. The general approach to this starts with filtering of the DC part of the signal and then further filtering the AC signal to remove unwanted components like electromagnetic noise and current spikes from the motor's starting. A study made by Testa et al., suggests the use of high bandpass filter, which filters out all frequencies that are not between a specific cutoff frequency "ω_c", and a π filter for removing the DC component, and the use of subsequent tunable bandpass filter, switched capacitor filter and pulse check and correction procedures to condition the circuit for the microcontroller (Ghosh et al., 2018). Further recent studies have suggested the use of hybrid computation techniques such as the use of fuzzy logic (Mellah et al., 2018) and cascade-forward neural networks for better estimation of speed and position (Zarchan, P., & Musoff, 2000), but their usability in automotive environments is yet to be tested.

5.4 Signal Filtering

Every signal contains some noise. To be able to separate the noise from the signal, a filter is used. There are many different types of signal filters. Based on number of measurements required, they can be divided in to two types: Infinite impulse response (IIR) filters, that use infinite number of measurements in a feedback system; and Finite impulse response (FIR) filters, that uses a finite number of previous measurements. There are also filters based on frequency response. Some of these have been described in the following subsections.

5.4.1 Kalman Filter

The Kalman filter is one of the most commonly used methods for filtering signals. It is an infinite impulse response filter based on linear predictions that optimize the least-squares method. The filter was introduced in 1960 by R. E. Kalman in the article "A New Approach to Linear Filtering and Prediction Problems." According to Zarchan and Musoff, the Kalman filtering is probably the most important algorithm in use. It has been used in applications like the Apollo spacecraft, predicting fluctuations on the stock market, inside GPS devices, and many more (Biezen, 2015). The algorithm can combine measurements from one or multiple sensors with the predictions of a mathematical model, then use the information from all sources to find the optimal prediction. It can do so by taking the error in the estimate (E_{EST}) and the error in the measurements (E_{MEA}) into account, based on these errors, the Kalman Gain (KG) can be calculated. The KG is a fraction between one and zero that indicates how much the algorithm "trusts" the sensors or the mathematical model, the closer KG is to one, the more trust is put on the measurement. So, if $KG = 1$ is all information coming from the sensors and if $KG = 0$ all the information is based on the estimates from the mathematical model. Over time the KG will go closer and closer to zero since the information from the measurements tend to be jumpy and E_{MEA} tends to be fixed over time, meanwhile E_{EST} will be updated in each iteration (sen M. Kuo, Bob H. Lee, 2013). The process in the Kalman filter has two distinct parts, one estimation phase where the information from the mathematical model and the sensor from the previous time step (k-1) is used to perform an estimate for the current time step (k) and one updating phase. A weighted average is then taken from the estimate and the update depending on the *KG* and this is the updated state. The updated state is then sent as an output from the filter and used as input data for the next iteration, a flow chart that illustrates the process of a Kalman filter can be seen in Figure 10.

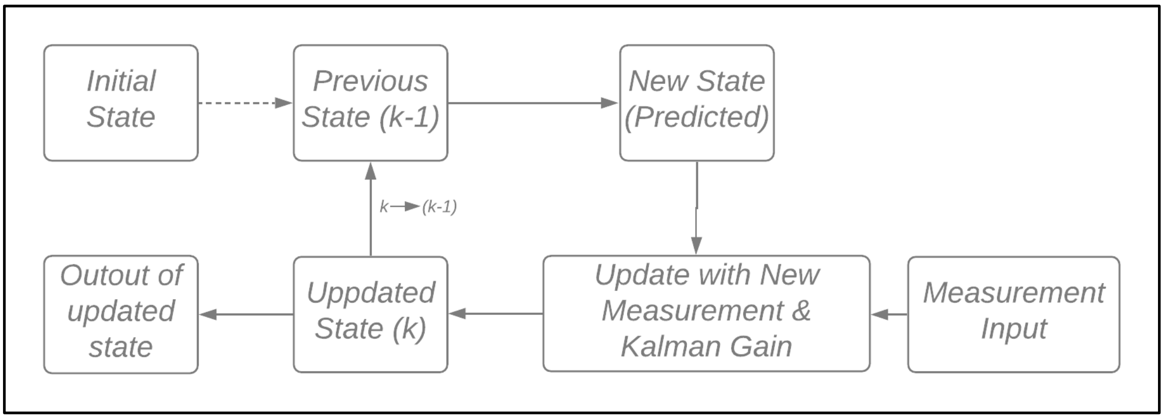


Figure 10 Flow chart illustrating the process of a Kalman filter

5.4.2 Finite Impulse Response (FIR) Filter

The fundamental difference between an IIR filter and a FIR filter is, as mentioned earlier, that the FIR filter only relies on a finite number of previous measurements. This number of measurements is denoted as "N" or "L" in literature and is called the memory size, the horizon or the length of the filter. For a filter with the length L is the order *L-1* since that is the number of zeros. As L increases, the noise suppression improves, but uncertainties in the model parameters will have a more significant effect on the results. So, it is vital to select a value of L that is suitable for the application. There are four different types of FIR filters, depending on if the length L is an even or odd number and what type of symmetry the coefficients have. A schematic diagram of a symmetric FIR filter with an even length can be seen in Figure 11. Since an FIR filter uses the previous measurements stored in the memory and not a feedback loop, these filters are more responsive, which gives a shorted delay between the input and the output signal. Not using a feedback loop facilitates the implementation of the filter and guarantees that the filter is stable at all times. Unfortunately, this also requires a bigger memory and a higher processing power, which often can be a limiting factor. (sen M. Kuo, Bob H. Lee, 2013)

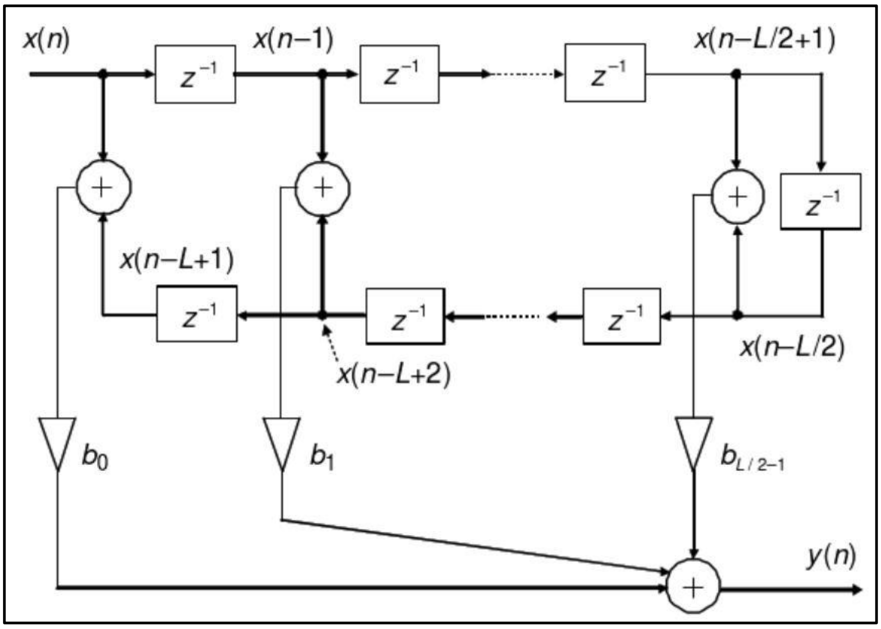


Figure 11 Schematic diagram of a symmetric FIR filter with the length L as an even number, z^{-1} is the tapped-delay, x(n) is the input signal, and y(n) is the output signal (Watters, 2015)

5.4.3 RC Filters

RC-filter stands for Resistor Capacitor filter, they consist of a resistor and a capacitor. These are the most fundamental type of frequency signal filters. Based on the configuration, RC filters can act as low pass, high pass, bandpass or band stop filters. By selecting the resistance and capacitance values is the cutoff frequency determined. The cutoff frequency is specified above or below which the filter will be active. A low pass filter only allows signals of frequency below the cutoff frequency to pass through. A high pass filter works in the opposite way and only allows signals of frequency higher than the cutoff frequency to pass. If a high pass and a low pass filter is combined can a bandpass filter be created, which only allows signals that lie between two cutoff frequencies to pass. In contrast, a band stop filter works in the opposite way an only allows signals that lie outside the two cutoff frequencies to pass through. Figure 12 illustrated how low, high, bandpass and band stop filters work.

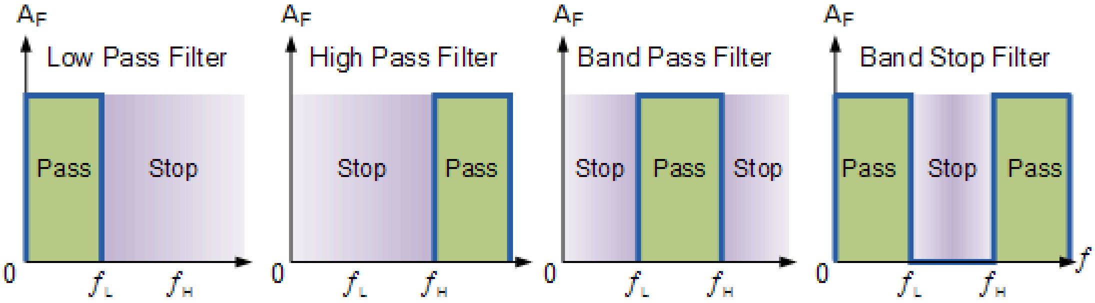


Figure 12 Range of frequencies allowed to pass through different filters (Storr, 2014)

5.5 Communication Protocols

In order to optimize the performance of the anti-pinch system, information from other sensors in the vehicle is needed. Parameters which are determined by other systems and are needed for the anti-pinch system are for example the ambient temperature and the speed of the vehicle. To obtain this information, a way for the system to communicate with the main control unit (MCU) is needed. The two most common protocols used in automobiles for this type of communication are CAN and LIN, which are explained further below.

5.5.1 CAN

Controller Area Network (CAN bus) is a serial communication protocol where communication happens in the form of messages. It is used for real-time control of many essential functions in automobiles, for example, for power train control, stability control, On-Board Diagnostics (OBD-II) based vehicle diagnostic standard, etc. The development of CAN was started by Robert Bosch GmbH. (*History of the CAN technology*, n.d.)

5.5.2 LIN

Similar to CAN, Local Interconnect Network (LIN) is also a serial communication protocol but is mainly used for low-cost automotive applications. It allows the creation of distributed electronic networks which are ideal for simple components, such as actuators and sensors (*LIN Steering Group*, n.d.). It was developed by the LIN Steering Group till Specification 2.2.A, after which LIN was made a part of the ISO 17987 Part 1-7 standard. The use of LIN is widespread in parts like light control panels and panel switches. (Rylander & Wallin, 2003)

6 Concept Generation

In this chapter, the sub-functions that need to be solved have been identified. This is followed by presenting the suggested solutions to these sub-functions, putting these solutions into a morphological matrix and generation of concepts. Finally, the screening process of concepts that do not meet the requirements has been described.

6.1 Identification of sub-functions

The first step in the concept generation process is to identify which sub-functions the generated concepts would need to be able to perform in order to meet the requirements. For the anti-pinch system, 8 functions were identified and are listed in Table 7. The next step was to come up with sub-solutions to the identified functions. These sub-solutions would later be used in the morphological matrix.

Table 7 Identified functions required in the Anti-Pinch system

No	Identified function
A	Driving the system
B	Detecting the pinch with mechanical contact
C	Detecting the pinch without mechanical contact
D	Filtering signal
E	Logic check position
F	Logic speed of process
G	Logic reverse distance
H	Communication with main ECU

6.2 Morphological matrix

The primary purposes of the morphological matrix are to visualize and organize the concept generation phase. This is done by listing the identified functions and their solutions in a user-friendly manner. The format of the matrix makes it possible to generate a vast amount of concepts in a short time. Each row in the morphological matrix is dedicated to a function and its solutions. Since 8 sub-functions were identified for the anti-pinch system, the matrix will have 8 rows. In the first column, the functions are listed, and the subsequent columns contain the solutions. The concepts are generated by picking one solution from each row to get a complete concept consisting of 8 sub-solutions in the end.

By following this method, a lot of these concepts will be unfeasible, since not all sub-solutions can be combined with each other. Most of the sub-solutions have been explained in great depth in the chapter Technology Study. A short description of each sub-solution can be seen in Table 8, and the morphological matrix can be seen in Figure 13.

Table 8 Short explanation of solutions to identified function

<table>
<tr><th>A Driving the system</th><th>B Detecting pinch with mechanical contact</th><th>C Detecting the pinch without mechanical contact</th></tr>
<tr><td><b>A1</b> A BDC motor drives the system
<b>A2</b> A BLDC motor drives the system</td><td><b>B1</b> Hall Sensors are used to monitor the motor performance
<b>B2</b> Ripple Count is used to monitor the motor performance
<b>B3</b> Capacitive sensors are used to detect contact with objects
<b>B4</b> Ultrasonic sensors monitor the back side of the moving to detect changes in the speed
<b>B5</b> Infrared sensors monitor the back side of the moving to detect changes in the speed
<b>B6</b> No system used</td><td><b>C1</b> Ultrasonic sensors monitor the opening to detect foreign objects
<b>C2</b> Infrared sensors monitor the opening to detect foreign objects
<b>C3</b> Capacitive sensors detect electromagnetic fields surrounding body parts in the way for the moving object
<b>C4</b> Image processing to monitor the inside of the vehicle and to detect obstacles in the way for moving objects
<b>C5</b> LiDAR monitor the inside of the vehicle and to detect obstacles in the way for moving objects
<b>C6</b> No system used</td></tr>
<tr><th>D Filtering signal</th><th>E Logic check position</th><th>F Logic speed of process</th></tr>
<tr><td><b>D1</b> Kalman Filter
<b>D2</b> Finite Impulse Response (FIR) Filter
<b>D3</b> No filtering</td><td><b>E1</b> The data from the sensor determine the position of the moving object</td><td><b>F1</b> The moving object has a constant speed over the entire range of motion
<b>F2</b> The speed of the moving object changes in different areas of the range of motion</td></tr>
<tr><th colspan="2">G Logic reverse distance</th><th>H Communication with ECU</th></tr>
<tr><td colspan="2"><b>G1</b> Reverses to a fully open position
<b>G2</b> Reverses 20% of its range of motion
<b>G3</b> Stop without reversing</td><td><b>H1</b> Communication via CAN
<b>H2</b> Communication via LIN</td></tr>
</table>

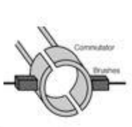
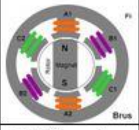
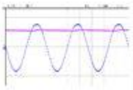
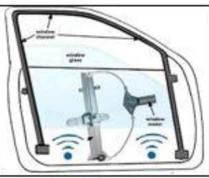
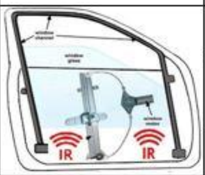
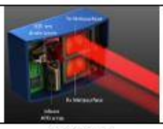
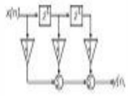
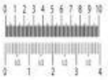
Morphological Matrix Anti-Pinch System						
Function Name	**Solution 1**	**Solution 2**	**Solution 3**	**Solution 4**	**Solution 5**	**Solution 6**
Driving the system A	 Brushed DC motor **A.1**	 BLDC motor **A.2**				
Detecting the pinch with mechanical contact B	 Hall sensors **B.1**	 Ripple Count **B.2**	 Capacitor sensor detecting contact **B.3**	 Ultrasonic to monitor back side **B.4**	 Infrared to monitor back side **B.5**	Nothing None **B.6**
Detecting the pinch without mechanical contact C	 Ultrasonic Sensor **C.1**	 Infrared sensor **C.2**	 Capacitor sensor detecting on distance **C.3**	📷 Camera **C.4**	 LiDAR **C.5**	Nothing None **C.6**
Filtering signal D	 Kalman filter **D.1**	 FIR filter **D.2**	Nothing None **D.3**			
Logic check position E	 Sensor data **E.1**					
Logic speed of process F	 Constant speed **F.1**	 Position dependent**F.2**				
Logic reverse distance G	100% Whole way **G.1**	20% Certain % of motion range **G.2**	STOP Stop, no reverse **G.3**			
Communication with main ECU H	CAN CAN **H.1**	lin LOCAL INTERCONNECT NETWORK LIN **H.2**				

Figure 13 Morphological matrix with solutions to different sub functions

6.3 Generated Concepts

From the morphological matrix, in theory 2592 concepts could be generated. In order to reduce the number of concepts to a more manageable level, the feasibility of combining the sub-solution was discussed. From that discussion, 17 feasible and unique concepts were generated. These concepts were given the names Concept 1 through Concept 17 and they are listed in Table 9 with a short description of which solutions they utilize.

Table 9 The 17 concepts taken from the morphological matrix with short explanations of how they work

Nr	Combined Solutions	Description
1	A1-B1-C6-D3-E1-F1-G1-H2	Most basic concept of all, uses hall sensors to monitors a BDC motor without any filtering. Constant speed through the entire range of motion, which allows for a much simpler motor controlling function. Uses LIN for communication
2	A1-B1-C6-D3-E1-F2-G1-H2	Bit more complex than 1, uses position dependent speed to reduce inertia in some areas for faster stop
3	A1-B2-C6-D3-E1-F1-G1-H2	Most Basic Ripple Count Concept similar to 1 but uses ripple count instead of hall sensors
4	A1-B2-C6-D3-E1-F2-G1-H2	Bit more complex than 3, uses position dependent speed to reduce inertia in some areas for faster stop.
5	A1-B1-C1-D1-E1-F2-G2-H1	Ultrasonic sensing for detecting pinches, hall sensors for position control uses position dependent speed and CAN to communicate with ECU
6	A1-B3-C3-D1-E1-F2-G2-H2	Capacitive sensors for detecting objects on distance and with contact, Kalman filtering, position dependent speed and LIN for communication
7	A2-B6-C5-D3-E1-F2-G3-H1	BLDC motor for high performance and LiDAR system to monitor the inside of the vehicle, stops before mechanical contact so no reverse needed, CAN for communication
8	A2-B2-C1-D1-E1-F2-G2-H1	BLDC motor combined with ripple count and ultrasonic sensing. Kalman filter combines the inputs from both sensing methods, position dependent speed and CAN
9	A1-B2-C1-D1-E1-F2-G2-H2	Same as 8 but uses a BDC motor as a cheaper alternative
10	A1-B2-C6-D1-E1-F1-G2-H2	Similar to 3 but uses a Kalman filter and only reverses certain % of the range of motion in case of a pinch detection
11	A1-B6-C4-D3-E1-F1-G3-H1	BDC motor, Camera for direct monitoring of vehicle, no filtering and stops no reverse if an object is in the way without making contact uses CAN for communication
12	A1-B4-C6-D3-E1-F1-G1-H2	BDC motor, uses ultrasonic to monitor back side of moving object no filtering or direct sensing, constant speed all the way and reverses % of range of motion if pinch, LIN communication
13	A1-B4-C1-D1-E1-F2-G2-H1	Similar to 12 but adds ultrasonic sensors for direct sensing and a Kalman filter to combine the sensors, position dependent speed and CAN
14	A1-B5-C6-D1-E1-F1-G1-H1	Infrared sensors monitoring back side of moving object, no direct sensing, Kalman filter is used and constant speed, reversing the full distance if pinch occurs and CAN for communication
15	A2-B5-C6-D3-E1-F1-G1-H2	Similar to 14 but uses a BLDC motor and no filtering
16	A1-B6-C5-D2-E1-F2-G3-H2	Anti-pinch will just be a function in a larger system that monitor the entire inside of the vehicle using LiDAR, no reverse is needed since it stops before contact. Signals are filtered using FIR and communication is done via LIN
17	A1-B6-C1-D2-E1-F2-G3-H2	Similar to 16 but uses ultrasonic sensors instead to monitor the inside of the vehicle

6.4 Concept Elimination

The next step in the development process was to screen the concepts that were generated using the morphological matrix against the demands in the customer needs list. For this, an elimination matrix was used, which allows for an unbiased and structured way of evaluating concepts.

6.4.1 Elimination matrix

The elimination matrix is manly used to remove concepts that do not meet the requirements or demands placed on them. Concepts also get removed if they don't solve the main problem, aren't realizable, are too expensive, dangerous for the user or if they don't fit the company's portfolio. The elimination matrix is considered to be a strong filter that removes many concepts. The concepts taken into the matrix are listed in a column, for each of the listed elimination criteria is the concept given a score; a (+) if it passes, a (-) if it fails, a (?) if more information is needed or a (!) if the specifications should be checked. After all concepts are scored, the concepts with a (-) were eliminated and more information was gathered about concepts with (?). (Almefelt, 2019)

The elimination matrix can be seen in Appendix B, below follows a summary of the results from the matrix.

All 17 concepts from the morphological matrix were taken into the elimination matrix, out of these were 9 eliminated. The eliminated concepts were; 5, 7, 8, 9, 11, 13, 14, 15 and 17, all eliminated concepts except 7 had a (-) on *is realizable,* and all except 11 and 17 had a (-) on *fulfills all demands,* in addition to this did concept 7, 11, 15 and 17 have a (-) on *have a reasonable cost.* More information (?) was needed to determine if concept 14 would *fulfill all demands* and be *realizable*. A more detailed explanation of why the individual concepts got deleted follows.

Concept 5, 8, 9 and 13 got eliminated mainly due to the complication that arises due to that the ultrasonic sensors, which the concepts are based on, needs to be located on a different location than the rest of the module provided by the supplier. This is troublesome since the supplier has design responsibility for one module and is not allowed to affect anything outside of it. In addition to this is the ultrasonic sensor itself a problem. The sensors should be mounted in a way, so that they can monitor the opening of for example the window to detect objects that are in its path. The sensor itself shouldn't risk being in the way for the window which could make it hard to combine it with the existing hardware in the vehicles.

Concept 7 and 15 got eliminated due to the unnecessary extra cost that is connected to using a BLDC motor instead if a BDC motor. BLDC motors are often good motors with long service life and almost no maintenance but the loads on the motors are relatively low in these applications and they are more expensive that BDC motors.

Concept 11 was eliminated based on that processing images is a heavy process and expensive, a camera is also sensitive to light conditions. Furthermore, having a camera monitoring the inner compartment of the vehicle at all times could bring some passenger privacy issues.

Concept 17 was eliminated since ultrasonic is susceptible to disturbances, which would reduce the reliability in such an open environment.

Concept 14 and 15 required more information before they could get any score for the criteria *fulfill all demands* and *are realizable*. This was due to some uncertainties in the performance of infrared sensors available in the market, it was found out that the available sensors would not be suitable for continuous monitoring of a moving object. So, both concepts got eliminated.

In addition to this was concept 3 and 10 combined due to their similarities leaving 7 concepts for the next step in the process. The remaining concepts and their paths in the morphological matrix can be seen in Appendix C.

6.5 Concepts Proceeding from the Elimination Matrix

Out of the seven concepts that remain in the process after the elimination matrix are two based on Hall sensors, two are based on ripple counting, one on capacitive sensors, one on indirect sensing using ultrasonic sensors, and one based on LiDAR technology. This section will give a more detailed description of these concepts. The sub solutions that together creates the concepts are listed in Table 9 in the Generated Concepts section above, in order to quickly give an understanding on what sub-solutions that are the same in the remaining concepts and which that differs are the paths of each concept in the morphological matrix shown in Appendix C. From the figure, all remaining concepts use a brushed DC motor for driving the system and LIN for communication, but apart from these sub-functions, all other sub-functions have very different sub-solutions.

Concept 1

Just as stated in Table 9, is this the simplest of all concept. The entire concept is based on that a number of hall sensors are placed in close proximity to the BDC motor poles or the motor shaft. The sensitivity of the system depends partly on the number of hall sensors and partly on the number of poles in the motor. The hall sensors detect when a motor pole is passing close to them and since the number of poles in the motor is known can the system keep track of how many times the motor has rotated. For example, if the motor is used to move a window up and down and it is known that when the motor makes one revolution is the window raised 1 mm, if the motor makes 100 revolutions is the window raised 100 mm. By knowing when the window is fully closed or fully open, and then counting the number of revolutions the motor has done can the system control the position of the window, which is a necessity for an anti-pinch system.

By measuring the time between two sensor inputs can also the rotation speed be calculated. The speed will naturally change over the range of motion for the window in the example due to reasons like variation in the contact with the rubber seal surrounding the window. But if an obstacle like an arm would come in contact with the upper edge off the window would the speed decrease unnaturally. If the speed becomes lower than a pre-determined value will the system act like there is something blocking its path and a pinch is detected, the window will then stop and reverse the full distance. The signals from the Hall sensors are so stable that no filtering is needed.

Concept 2

Concept 2 is based on the same principles as concept 1 for position control and for detecting pinches. The difference lies in that for concept 2 is the motor speed decreased in regions where there is a high risk for pinching accidents. Decreasing the speed lowers the inertia, which makes it easier to stop, and as a result, the pinching force is lowered. A PWM signal is used to regulate the speed of the motor.

Concept 3

Concept 3 uses the ripple counting method for both position control and for detecting pinches instead of hall sensors that the concept 1 and 2 does. By using ripple count, more inputs per motor revolution can be given, this means in practice that a change in speed can be detected much quicker and the system can detect a pinch situation earlier which reduces the pinch force. Except from that is the position control and anti-pinch system working in the same way as for a system based on hall sensors, it just uses the ripples instead of sensor inputs.

Concept 4

Concept 4 is a combination of concept 2 and 3, it utilizes ripple counting technology and speed control. This gives a very exact system with low pinch forces since the ripple counting method allows for exact control and reducing the speed will enable quick reactions by the system.

Concept 6

Concept 6 relies on a completely different method for detecting pinches compared to the first four concepts. It is the first direct system, so instead of monitoring the motor in order to detect unnatural decreases in speed, a capacitive sensor is inserted into the seal on the closing edge of the windows or on the sunroof. The capacitive sensors can detect a pinch before there is any physical contact between, for example, the window and a finger. This is possible since the sensors detects changes in the electrical field surrounding the sensor. If something would enter this field, like a jacket, or if there is another type of object blocking the path, a pinch is detected. Capacitive sensors are excellent for an anti-pinch system, but they cannot be used for position control. This means that hall sensors or ripple counting is needed to determine the position of the window or sunroof. In order to combine the sensor inputs from the capacitive sensors and the sensors used for position control a Kalman filter is needed. The placement of a seal with inserted capacitive sensors can be seen in Figure 14.

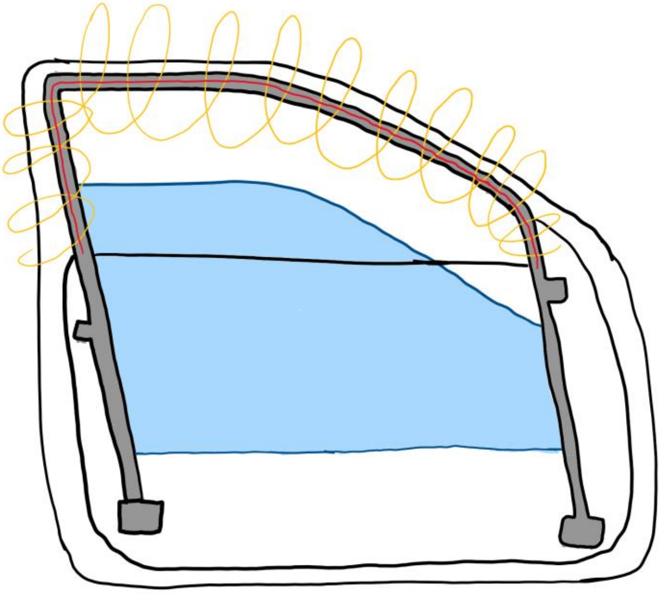


Figure 14 A sketch illustrating the use of capacitive sensors (in red) and their electric field (in yellow)

Concept 12

Both the position control and the anti-pinch system of concept 12 relies on indirect sensing using ultrasonic sensors. For the windows is the ultrasonic sensor places inside of the door and monitor the bottom side of the window. By doing so can the distance to bottom edge of the window and the speed with which it opens, or closes be determined. This system is relatively similar to the first four concepts with the difference that it uses inputs from an ultrasonic sensor instead of input from a hall senor or ripples from the motor. The ultrasonic sensor can give even more exact readings then the ripple counting method, but it has a big disadvantage when it comes to the placing of the sensor. For the window is there a possibility to place the sensors inside the door, but for the panel and the curtain in the sunroof is the situation different and it can be hard to find a good location for the sensor. Figure 15 shows an illustration of how the ultrasonic sensor could be places inside a door.

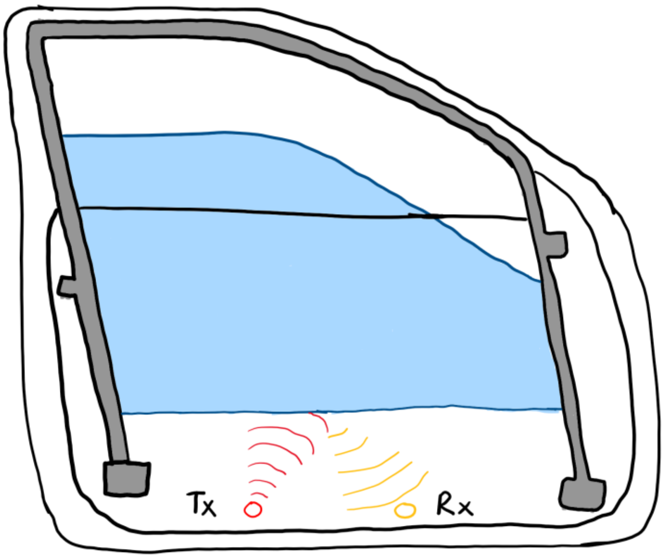


Figure 15 A sketch illustrating the use of ultrasonic sensors to indirectly measure a pinch situation

Concept 16

Concept 16 differentiates itself from all the other concepts, instead of just monitoring one window is two sensors placed in the front of the interior compartment. From that position can the LiDAR sensors create an 3D image of the inside of the vehicle and monitor all activity. This includes determining the position of the sunroof and all the windows, if one of the windows would be closing when a passenger have an arm out through the window is this seen by the system and the window is stopped before it can pinch the arm. This system is much more advanced than the other but can also be used for other functionalities than just the anti-pinch system and the position control of the windows and sunroof. A sketch of what the interior of the vehicle would look like from the perspective of the LiDAR sensors can be seen in Figure 16.

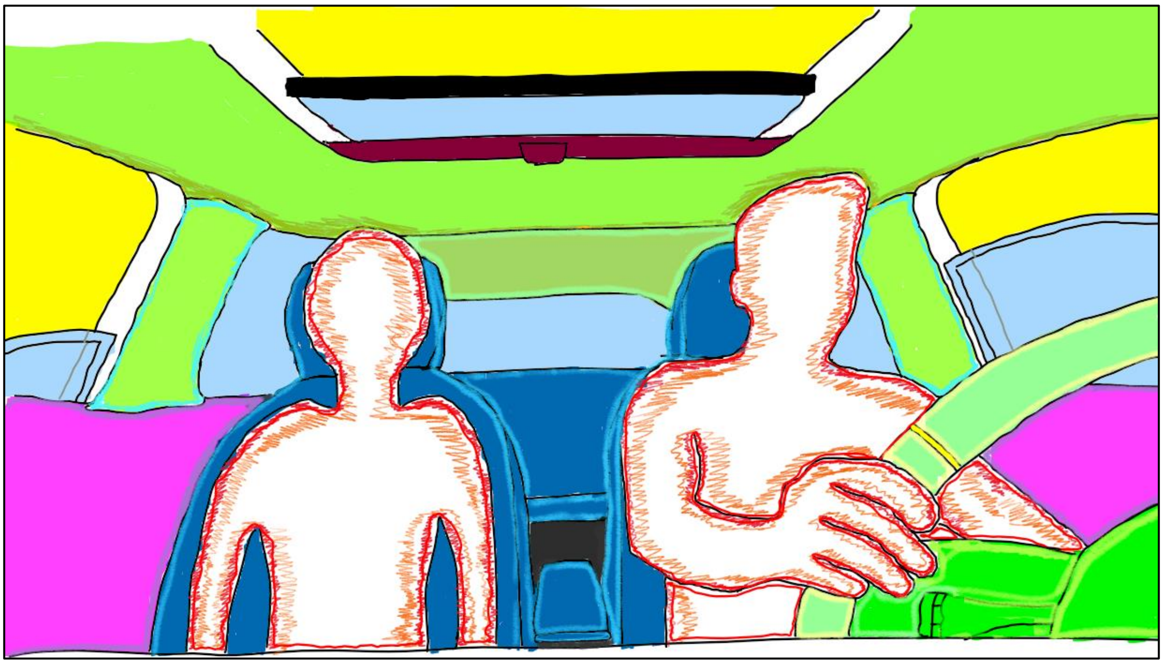


Figure 16 The sketch illustrated how the interior compartment of a vehicle would be seen by a LiDAR sensor

7 Prototyping

In this chapter, the prototyping efforts conducted in this project have been elaborated. Starting with the quick and dirty prototyping done early on in the project, and the continuation of the development work done for concepts 1, 2, 3 and 4. The hardware used and the process of developing the software is also explained in detail.

The prototyping phase took up most of the time in this project, a lot of things can be learned in theory but being able to implement the theory in a product or prototype adds much more values. The best scenario would be to build prototypes of all the 7 remaining concepts and then test them in the same way so that they can be compared in a fair manner. The time limitations in this project makes it impossible to build prototypes of all concepts, so the prototyping effort will be focused on concepts 1, 2, 3 and 4.

7.1 Early Prototyping Quick & Dirty

An early stage prototype was created using a LEGO EV3 programmable robotics kit. The prototype uses a motor to move a plate of LEGO blocks in a translating motion, similar to the motion of the sunroof panel and curtain. Two different approaches to anti-pinch systems were implemented. The first approach monitored the speed of the motor, like a system based on hall sensors would do. The second system measured the current that the motor required. Both systems were functional, they stopped and reversed the panel when an obstacle was detected. Due to LEGO’s user friendly block programming, this simple but functioning prototype could be built and tested in approximately 3 hours to translate theoretical knowledge into practice. The LEGO EV3 brick and the LEGO panel can be seen in Figure 17.

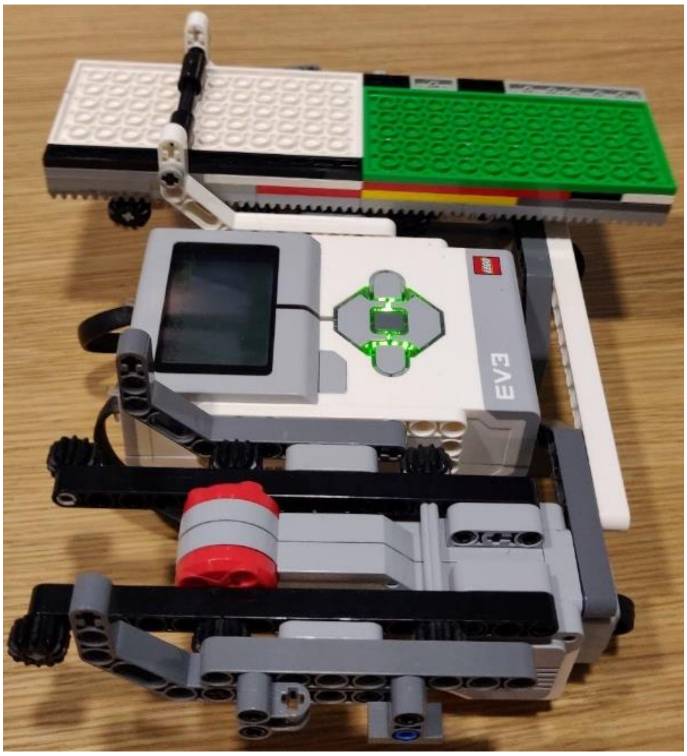


Figure 17 LEGO EV3 system used for early prototyping in order to increase the understanding of how a system would work in practice

7.2 Prototyping Concept 1

The prototypes of the four concepts rely on similar hardware and software set ups. The prototyping started with concept 1. As a result, a lot of functionalities and circuitry developed for concept 1, could be carried over to the prototypes of the other concepts. Because of this, a lot of the common basic functionalities are explained in this section.

7.2.1 Prototyping Hardware

A cut off section of the roof from a standard Volvo S60 was used as the base for building the prototypes. This section was cut off on the A, B, C and D pillars, leaving a complete roof without any interior parts. This allowed for quick and easy access to the BDC motors mounted on the roof for operating the panel and the curtain beneath it. The roof was mounted on a trolley to secure it from falling and to allow easy access and mobility. The cut off section of the roof can be seen in Figure 18.

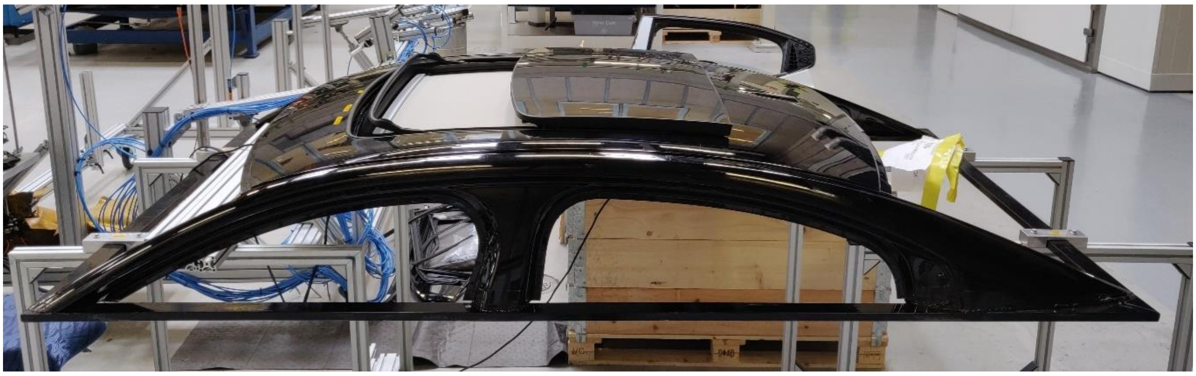


Figure 18 The cut off section of the roof, including sunroof panel and curtain mounted on a trolley

The two brushed DC motors on the roof were from the manufacturer Nidec. These are standard for the sunroof on Volvo vehicles. The motors have an 8-pole ring magnet mounted on the shaft and two hall sensors integrated in the motor. Both hall sensors are placed on the upper side of the motor as can be seen in Figure 19 and can be accessed via pins in the motor connections. The 8-pole ring magnet in combination with the two hall sensors gives 8 positive readings per revolutions, 4 from each hall sensors, this sets a physical limitation of how sensitive this system can become since it is not possible to increase the number of readings per revolution and thereby a quicker reaction. A diagram of the hall sensor signals from the two sensors for one revolution can be seen in Figure 20.

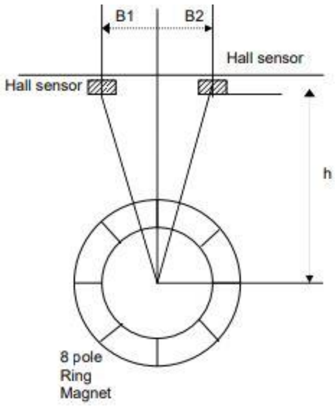


Figure 19 Positioning of the two hall sensors in relation to the 8-pole ring magnet

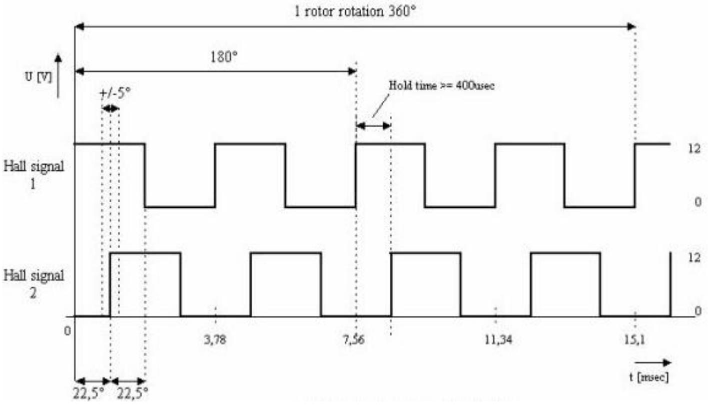


Figure 20 Diagram of the hall sensor inputs from the two hall sensors for one revolution in clockwise direction

In addition to the roof, a front and a rear door from Polestar 2 was used for developing the prototype for the power windows, as can be seen in Figure 21. To get access to the power window motors, the inner panels were removed from both doors. Figure 22 shows the inside of the front door without the inner panel so that the placement of the power window motor can be seen. No trolley or frame was built for the doors, instead they stood in EUR-pallets.

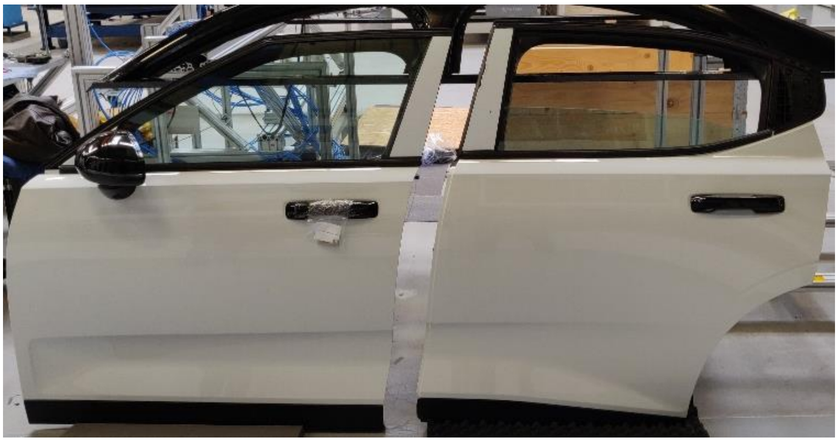


Figure 21 Front and rear doors of a Polestar 2

The motor situation looked a bit different for the doors compared to the roof. The hall sensors were positioned on the door's control board and then inserted into the motor as can be seen in Figure 23 and Figure 24 instead of being integrated in the motor. The hall sensors could not be accessed via the pins on the door control board because the signals are processed directly and never leaves the board. To get access to the hall sensors, connections were made directly on the board instead of connecting to a pin as for the roof motors. The wires soldered to the sensors can be seen in Figure 25.

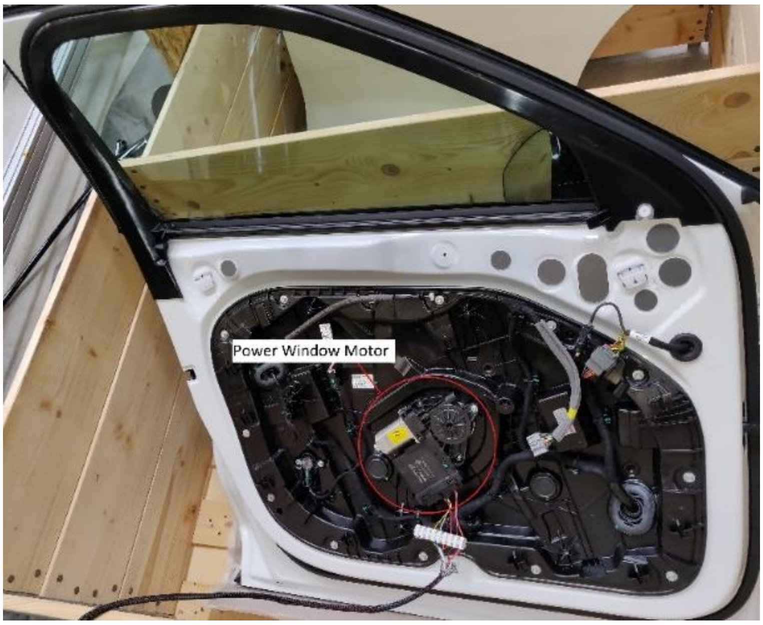
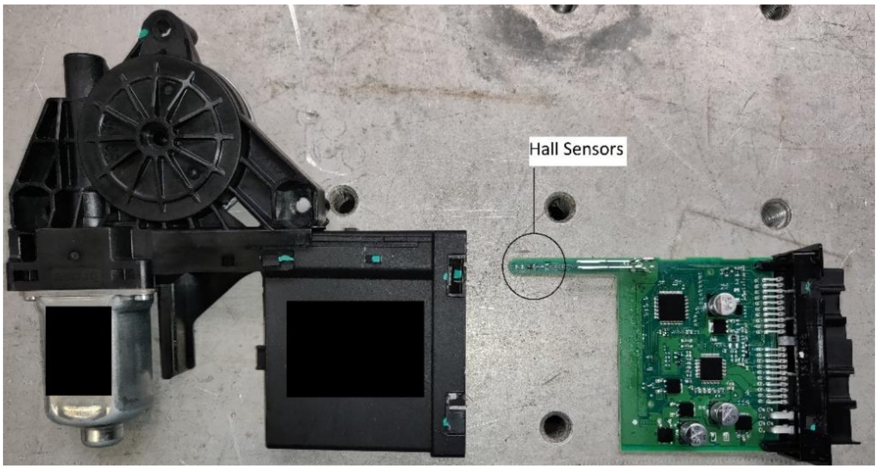


Figure 22 Inside of the front door showing the placement of the power window motor



[redacted]

[redacted]

Figure 23 The hall sensors are mounted on the MCU and then inserted into the motor so that they can get in contact with the poles on the ring magnet

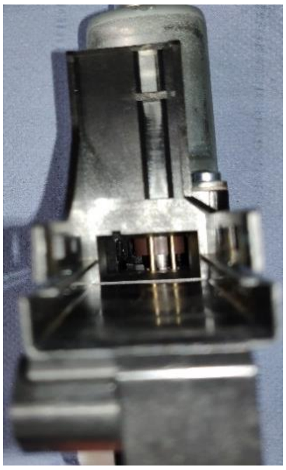


Figure 24 The opening in the motor where the hall sensors are inserted

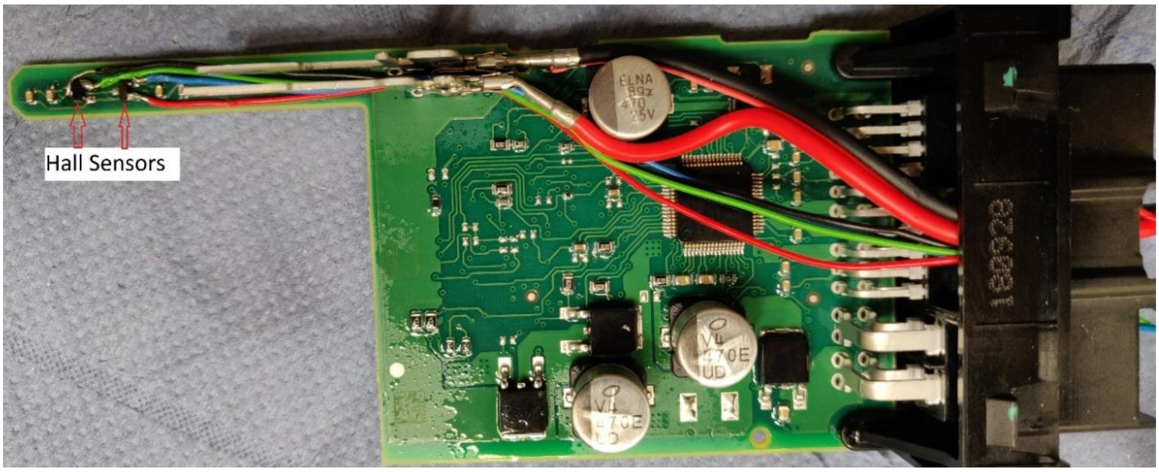


Figure 25 To get the hall sensors signals where connections soldered directly to the sensors on the MCU

To drive and control the motors, an electrical and an electronic system was set up. To provide the power to run the motors, a DC variable power supply was used in constant voltage mode at 12V. For processing the sensor input a microcontroller was needed. For reasons of simplicity and ease of use, an Arduino Mega 2560 was used for this purpose and to control the motor movement based on user inputs. A circuit diagram illustrating how the components, including the Arduino Mega, the power source, motors etc., in the electrical circuit are connected to each other can be seen in Figure 26.

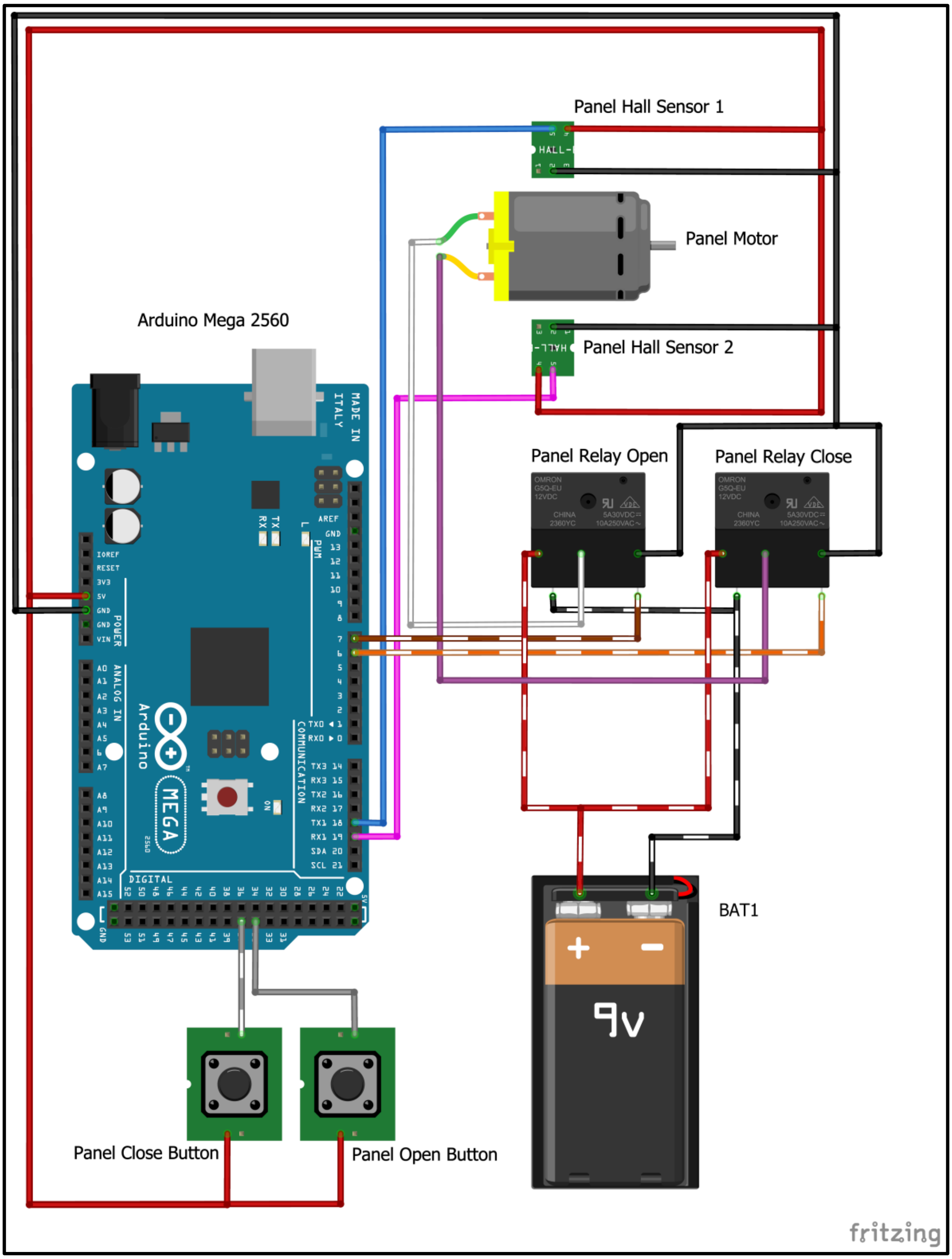


Figure 26 Circuit diagram showing concept 1 for the sunroof panel with the motor, hall sensors, Arduino Mega, control buttons, power source and relays

7.2.2 Prototyping process

A multistep prototyping process was carried out with the focus of implementing one required function in each step. The development work started with the sunroof panel. When all functions were implemented on the panel, the software code was copied and modified for the sunroof curtain, and then for the power windows.

7.2.2.1 Sunroof Panel Concept 1

The first step in the process was to establish position control. This means the ability to open and close the panel, determine the range of motion and the position of the panel in every instance. This was done by detecting the rising edges from the hall sensor signal to see how many inputs were received for a certain distance. Establishing position control is relatively complex for the panel, since the panel needs to tilt up before it can slide into its fully open position. During this lifting procedure, the performance of the system is different compared to other sections of the opening and closing operations where the panel is only sliding. The fully closed, tilted and fully open positions of the sunroof panel can be seen in Figure 27 to Figure 29. It was observed that during the tilting procedure, 197 signals arrive from the hall sensors, whereas over the remaining range of motion 1188 such signals were received giving a total of 1385 hall sensor inputs from a fully closed to a fully opened position. These hall sensor inputs are used to determine the distance of the panel.

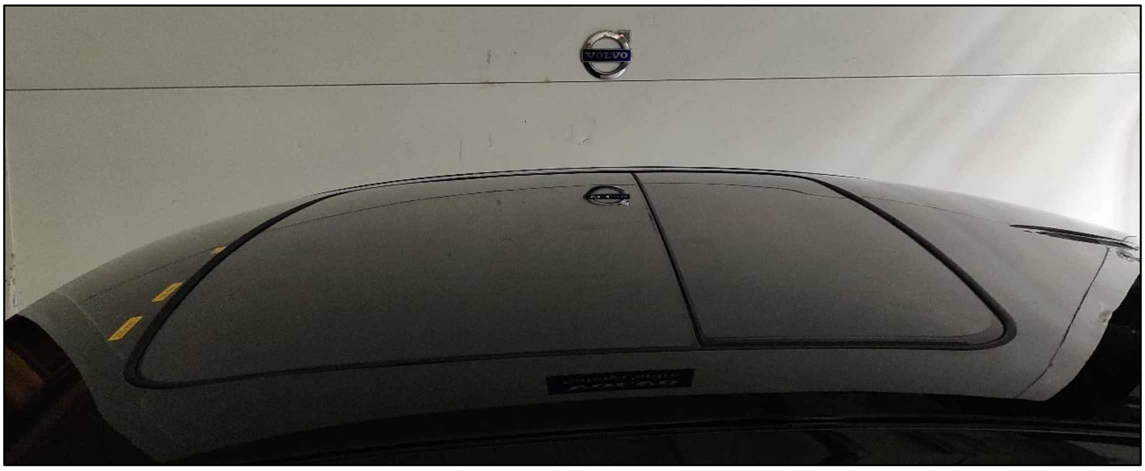


Figure 27 Sunroof panel in its fully closed position

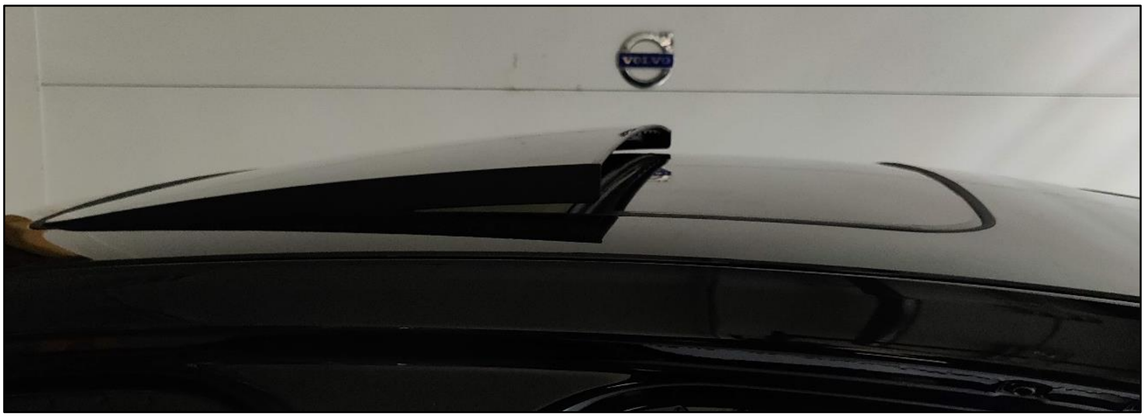


Figure 28 Sunroof panel in its tilted position

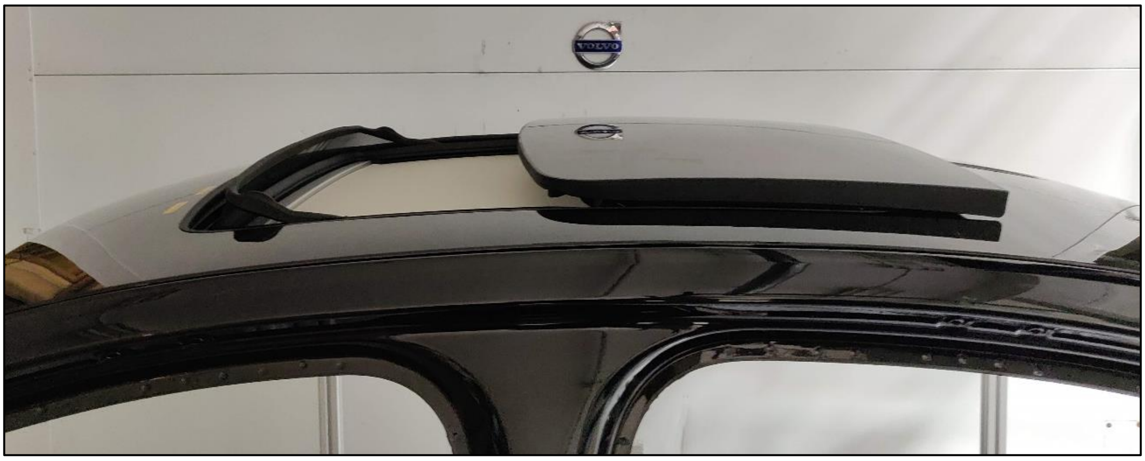


Figure 29 Sunroof panel in its fully open position

When the position control worked reliably, the next step was to make the system identify pinch situations. This started with using the hall sensor inputs to calculate the speed of the motor in every position of the panel. Since the sensors are not symmetrically located around the 8-pole ring magnet, four sensor inputs are needed to get somewhat uniform measurements. The time it took to get these four inputs is measured and used to calculate the rotational speed of the motor. The calculations of the rotational speed are done in two steps, first the absolute value of the time difference between the first inputs from hall sensor 1 and hall sensor 2 calculated and saved. The same calculation is then done for the next inputs from both sensors. Finally, the difference between the newer and older measurement is calculated as:

$$Time\ Per\ Revolution = abs(Time\ Difference\ New - Time\ Difference\ Old\ )$$

So, the variable name "Time Per Revolution" is a misnomer, since it only takes four measurements into account and not eight which would be needed for a full rotation, but it gives a good and simple explanation of what parameter the variable is supposed to convey. Thus, the variable "Time Per Revolution" is inversely proportional to the speed of the motor; the slower the motor rotates, the higher will the time per revolution be - as the time between the hall sensor inputs will be longer. An explanation of how this was done in the software is given in the Software Development chapter.

When the "Time Per Revolution" could be calculated in every position of the panel, ten consecutive test runs were made to see how the time per revolution varies over the range of motion and between different runs. These values were saved and plotted in MATLAB to illustrate the operation variance. These measurements can be seen in Figure 30.

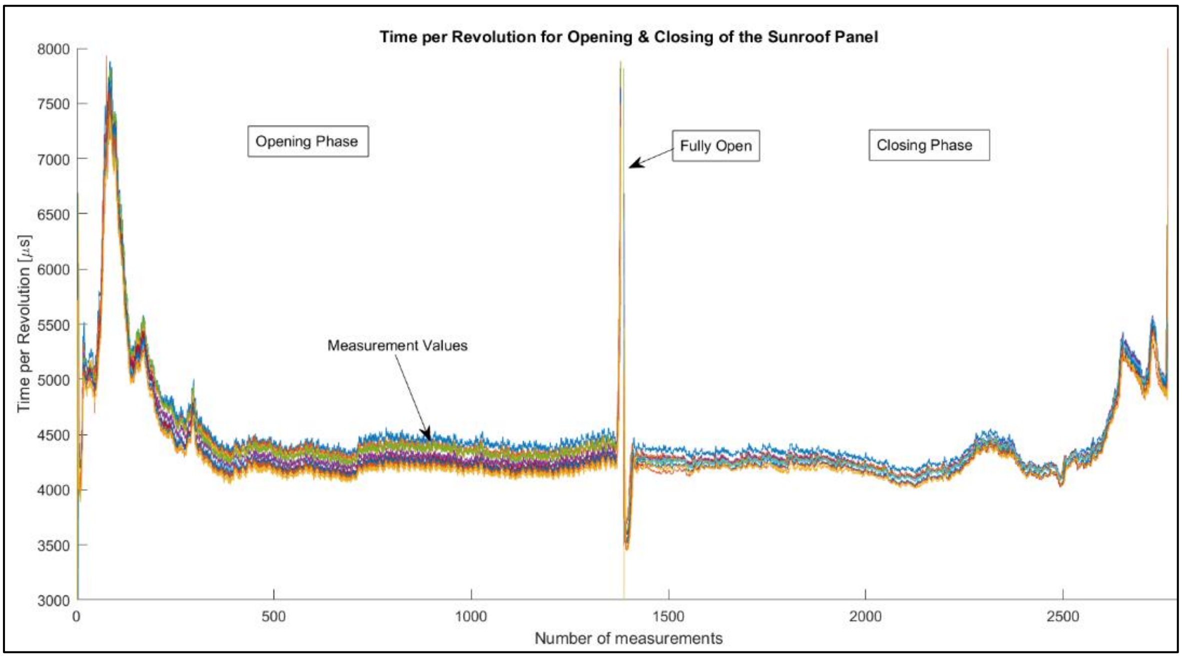


Figure 30 Time per revolution measurements for the sunroof panel. The spike in the middle shows the fully open position and the distinguishing regions in the beginning of the opening phase and in the end of the closing phase is the tilting

7.2.2.1.1 Threshold Curve

Once all values from the 10 test runs were recorded, these “Time Per Revolution” measurements were then used to determine the threshold values for the zones where the anti-pinch system should be activated. The threshold line is a curve fit created with the help of Microsoft Excel, that is raised above the measurement values. If the panel hits an obstacle, its speed will decrease leading to longer time interval between the hall sensor inputs. This in turn leads to the increase in the value of “Time Per Revolution”. If the “Time Per Revolution” increases to the extent that it intersects the threshold line, a pinch is detected, and the motor stops and reverses until the panel reaches its fully open position. The threshold curve and the ten measurements for the closing of the panel can be seen in Figure 31.

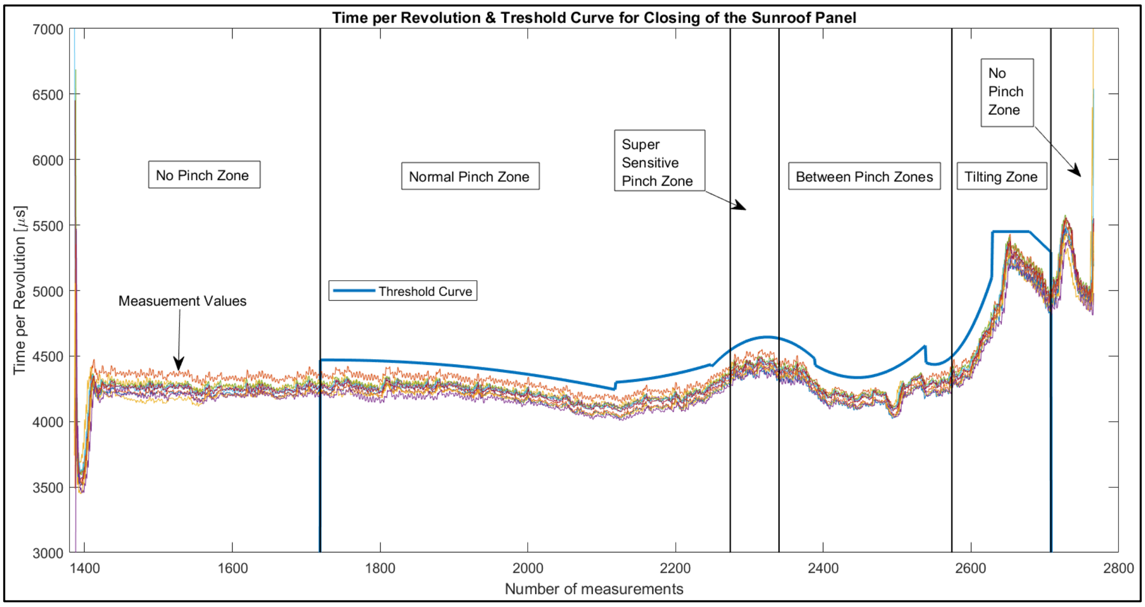


Figure 31 10 Time per revolution measurements for the closing phase of the sunroof panel, the threshold curve can also be seen for the parts where the anti-pinch system is active

Depending on the position of the panel, different amount of speed reduction will cause the time per revolution to pass the threshold line. For this, six different zones were defined. The zones can be seen in the plot in Figure 31, and all zones except the “Tilt Zone” are marked with lines A, B, C and D in Figure 32. Determining the required difference in amplitude between the time per revolution curve and the threshold curve was an iterative process. The amplitude of the threshold curve was set to an initial value, and then the corresponding pinch force at that amplitude was measured. If the pinch force was found to be too high, the amplitude of the threshold curve was lowered, and if the system had a tendency of detecting false pinches, the threshold curve was raised, and new pinch force values were gathered. This process was repeated until the pinch force was at a good level and no false pinches were detected.

In the “No Pinch Zone” before line “A” in Figure 32, the anti-pinch system is not active. The “Normal Pinch Zone” starts at line “A” that was 200 mm from the edge and ends at line “B”. In this zone, the spring stiffness required during the legal testing for FMVSS No. 118 is 20 N/mm. The “Super Sensitive Pinch Zone” lies between lines “B” and “C” which is the last 25 mm before the panel has covered the entire opening seen from the inside of the vehicle. In the “Super Sensitive Pinch Zone”, the spring stiffness used during testing is 65 N/mm. Resulting in that the panel can only move 1.5 mm before the pinch force reaches 100 N. The next zone is the “Between Pinch Zones”, between line “C” and “D”, this zone is unique since the opening that could be seen from the inside of the vehicle is already covered by the panel. But the panel will keep on moving for another 50 mm before the front edge has reached its final position seen in Figure 28. There is a possibility that a pinch situation occurs if the user, for example, is gripping the edge of the inner ceiling, and therefore the anti-pinch system is still active in this zone. The fifth zone is the “Tilt Zone”, where the front of the panel has already reached its final position, but the rear end is lowered down as can again be seen in Figure 28. Finally, there is one more zone where the anti-pinch system is deactivated, because the panel is less than 4 mm from the edge.

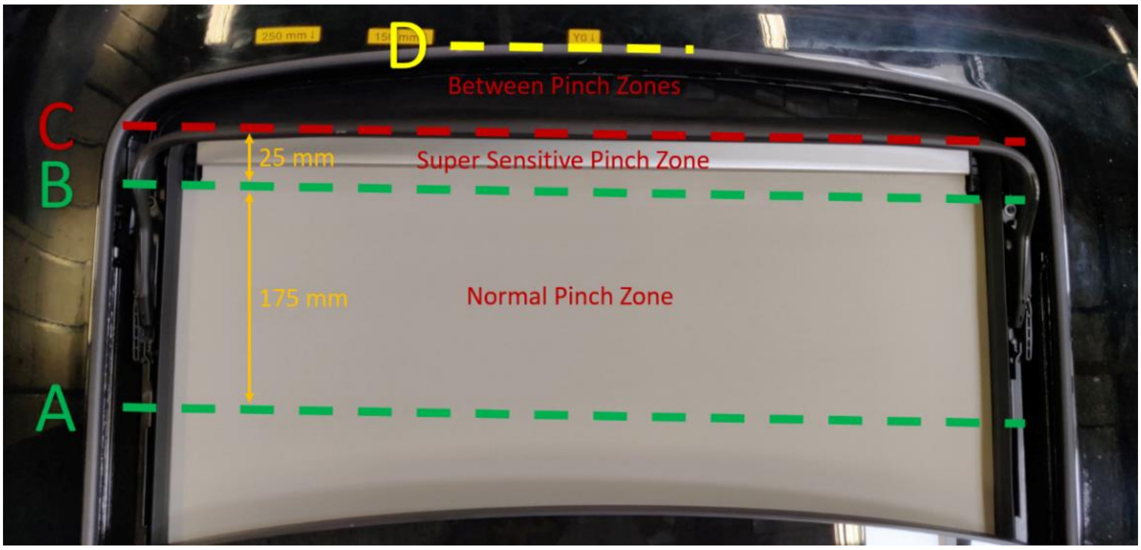


Figure 32 Line A marks the start of the “Normal Pinch zone”, the “Super Sensitive Pinch Zone” is between the lines B and C. At line C has the panel cover the entire opening that can be seen from inside the vehicle, but it will keep closing until it has reached line D before it starts to tilt down.

The closer the threshold curve is to the actual readings, the more sensitive the system will be towards pinch detections. This is a tradeoff between sensitivity and the risk that the system detects “false pinches” if the motor would slow down for any other reason than that there is an obstacle blocking its path. It is important to notice that these test values are recorded in lab conditions without any impact from vehicle vibrations, air resistance, extreme temperatures or other elements. These factors could force the lowering of the sensitivity in order to avoid false pinches.

7.2.2.1.2 Adaptive Threshold Curve

One way of avoiding “false pinches” caused by changes to the factors which are consistent throughout the entire closing phase, like ambient temperature, was to make the threshold curve adaptive in nature. Out of the ten consecutive test measurements shown in Figure 31, the first and the last of those measurements are separately shown in Figure 33. The only factor that causes the differences between these two measurements is the motor temperature. The change in motor temperature resulted in a difference in the amplitude of the time per revolution curve but not its shape. The measurements from the tenth test run are much lower than the ones from the first run. To cope with this difference in amplitude, reference values called “threshold base” are gathered from the last four hall sensor inputs before the panel enters the pinch zone. This is done every time the panel is closing. The amplitude of the threshold curve is adjusted based on these threshold base values, so that the curve has the correct sensitivity for every specific instance the panel is closed.

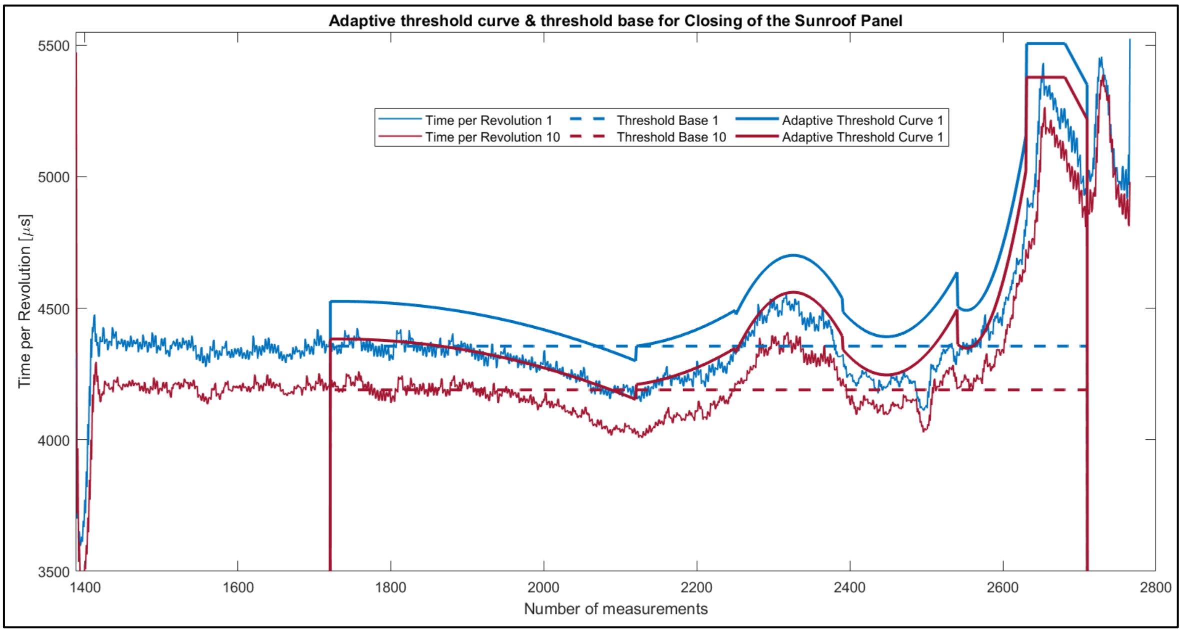


Figure 33 Measurement values from test run 1 and 10 with their respective threshold base line and threshold curve.

7.2.2.1.3 Hard Stop

One of the most important basic functions is the hard stop. It is used to prevent the drifting of the distance, in turn the threshold curve, and ensuring a safe operation. The hard stop function detects when the panel has reached its fully open or its fully closed position and stops the movement by turning off the current supply to the motor. This is done by checking if an open/close signal is given by the user and comparing the time difference between two hall sensor inputs to a pre-determined value. For example, if the closing button is pressed, the signals from the hall sensors come in at a regular basis until the panel reaches its fully closed position. Then it becomes impossible for the panel to move any longer and the motor will stop rotating. When the motor isn’t rotating, no signals will be received from the hall sensors. After 250 µs without any sensor input, the hard stop will turn the motor off and set the distance to 0.

The system functions in the same way for the opening procedure, and when a hard stop is detected, the distance is set to the fully open distance of 1385. Resetting the fully open and fully closed distance every time the panel reaches one of its end positions minimizes the drifting of the threshold curve due to varying numbers of hall sensor inputs that are received. If the panel is opened and closed several times in a row without the hard stop function, the threshold curve was found to be misplaced, causing false pinches or a low sensitivity.

7.2.2.2 Sunroof Curtain Concept 1

For the sunroof curtain that sits beneath the panel, the procedure of building the code required less effort since all functions like the hard stop and the adaptive threshold curve where already implemented on the panel. The panel code was just copied and variable names where change to fit for the curtain. Then the same procedure for determining a threshold curve was repeated. The major difference for the curtain is that there is no tilting phase which makes it much easier to control. Since it also has a longer range of motion than the panel, 2150 hall sensor inputs are received to reach the fully open position. The curtain sits on the inside of the roof and when it is opened is it rolled up on a cylinder behind the sunroof. The placement of the curtain, its motor and the roll-up structure are shown in Figure 34. The time per revolution for the entire range of motion and the threshold curve for the closing phase can be seen in Figure 35, and Figure 36.

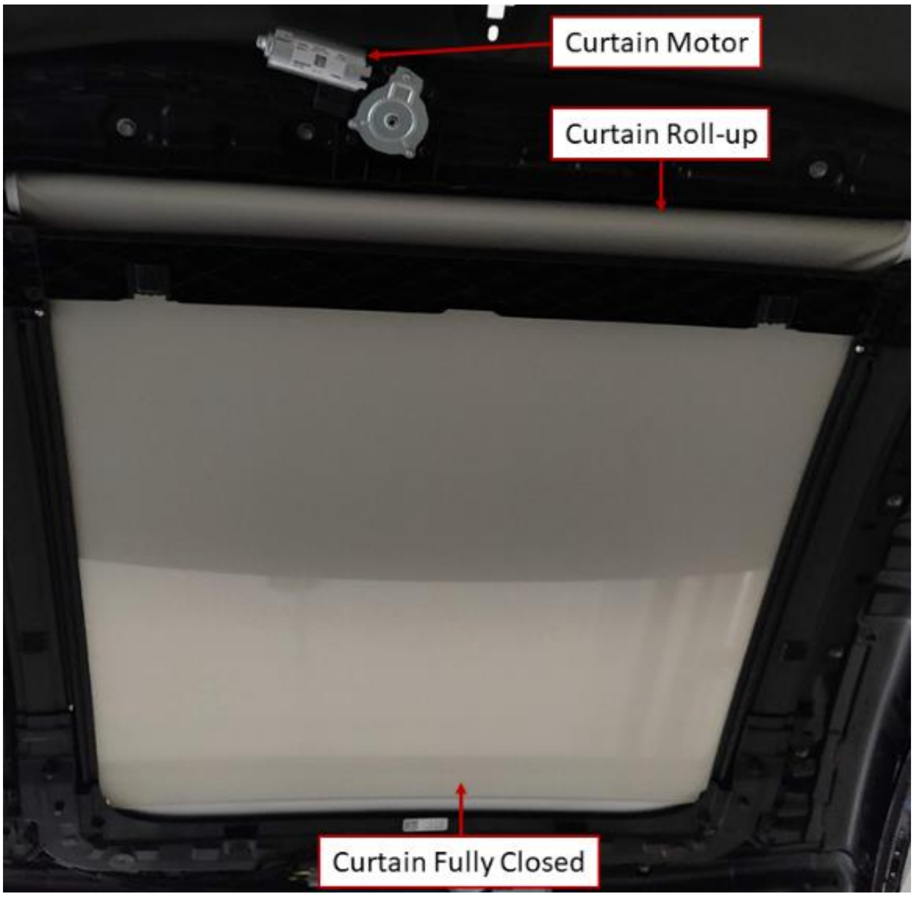


Figure 34 The placement of the sunroof curtain, its motor and the roll-up structure

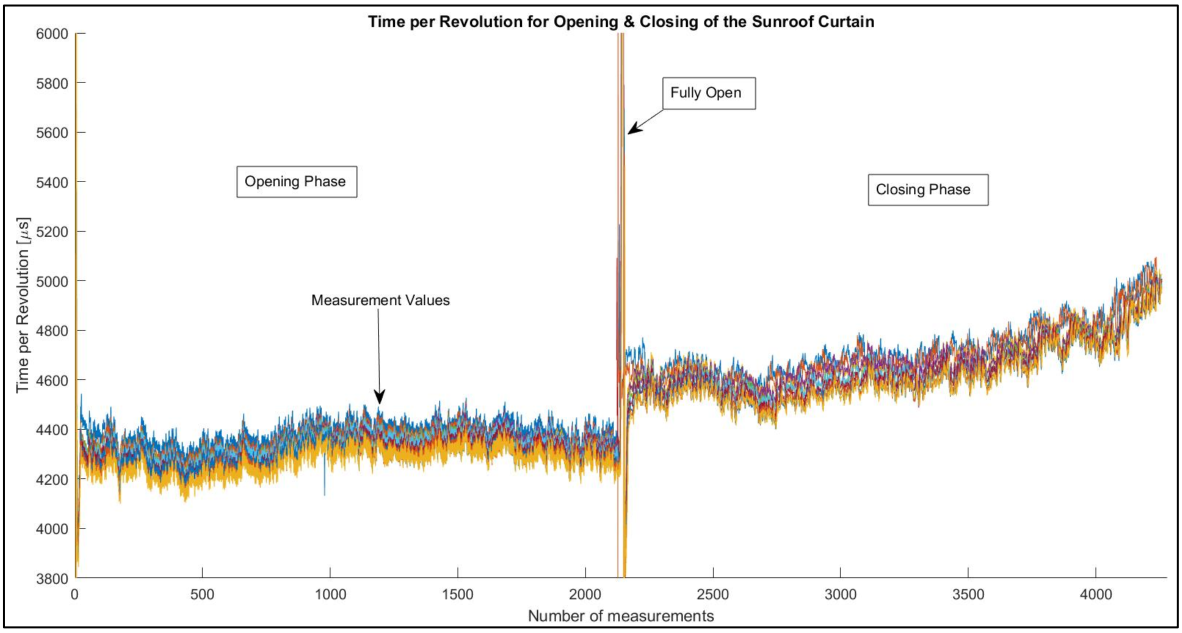


Figure 35 Time per revolution measurements for the sunroof curtain

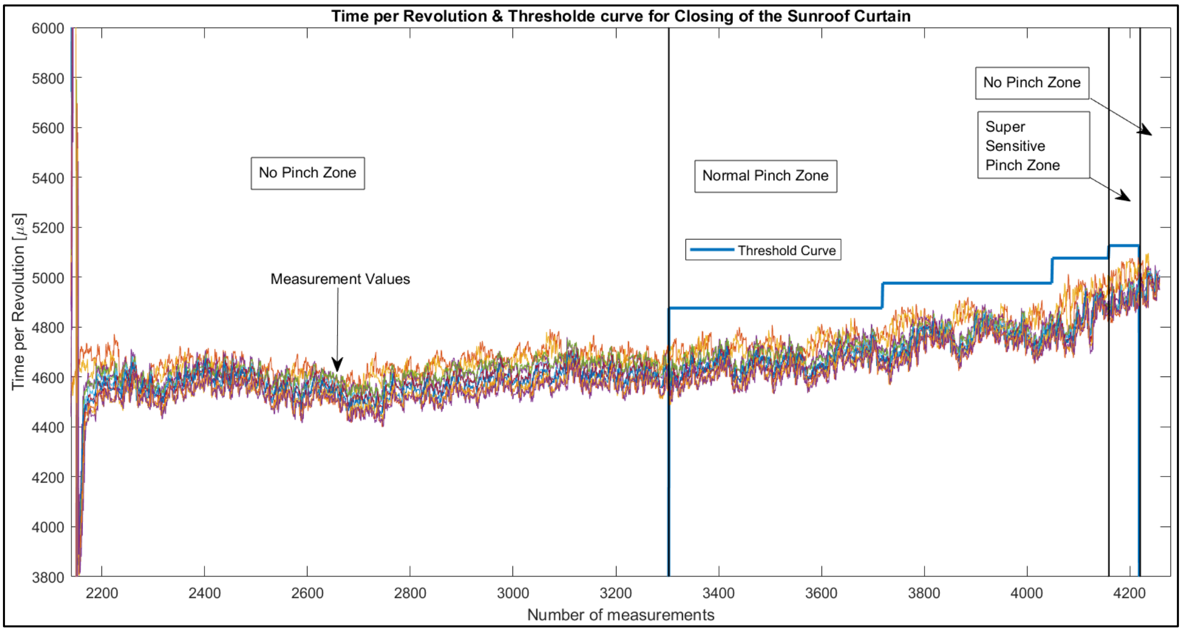


Figure 36 Time per revolution measurements for the closing phase of the sunroof curtain, the threshold curve can be seen for the parts where the anti-pinch system is active

7.2.2.3 Power windows Concept 1

For the power windows, the existing code with all functions was copied and modified in a similar way as for the curtain. The code that was built for the sunroof panel, worked for the window without any issues. However, some problems arose due to the improvised connections that were made on the door control board, as described in the Prototyping Hardware section and showed in Figure 25. The connections worked good for a little while testing the windows, but after opening and closing the windows multiple times, one of the motor’s hall sensor got damaged. Different reasons for why the sensors were “dying” were investigated, but after testing and destroying sensors on four different control boards, for both the front and rear doors, the investigation was canceled due to lack of time and new boards. Instead, it was accepted that only one hall sensor could be used for the window, which reduced the sensitivity of the system drastically. During the full range of motion 460 hall sensor inputs are received which is half compared to if the system would have both hall sensors functional. Since the same software was used for the window as for the panel, the only development effort, except creating the circuit, was to gather measurement values and create a threshold curve. The threshold curve was created using the same iterative process as described for the sunroof panel. Unlike the sunroof panel and curtain, the windows slide in the vertical direction. This affects the time per revolution measurements and the current consumption. As can be seen in Figure 37, there is a significant difference in the time per revolution measurements between the opening and closing phases. The naturally slower pace during the closing of the window is advantageous when it comes to getting a more sensitive anti-pinch system. A lower pinch force is caused by the reduced inertia. The closing phase consists of three zones; the no pinch zone, the normal pinch zone and the super sensitive pinch zone which are shown together with the threshold curve in Figure 38.

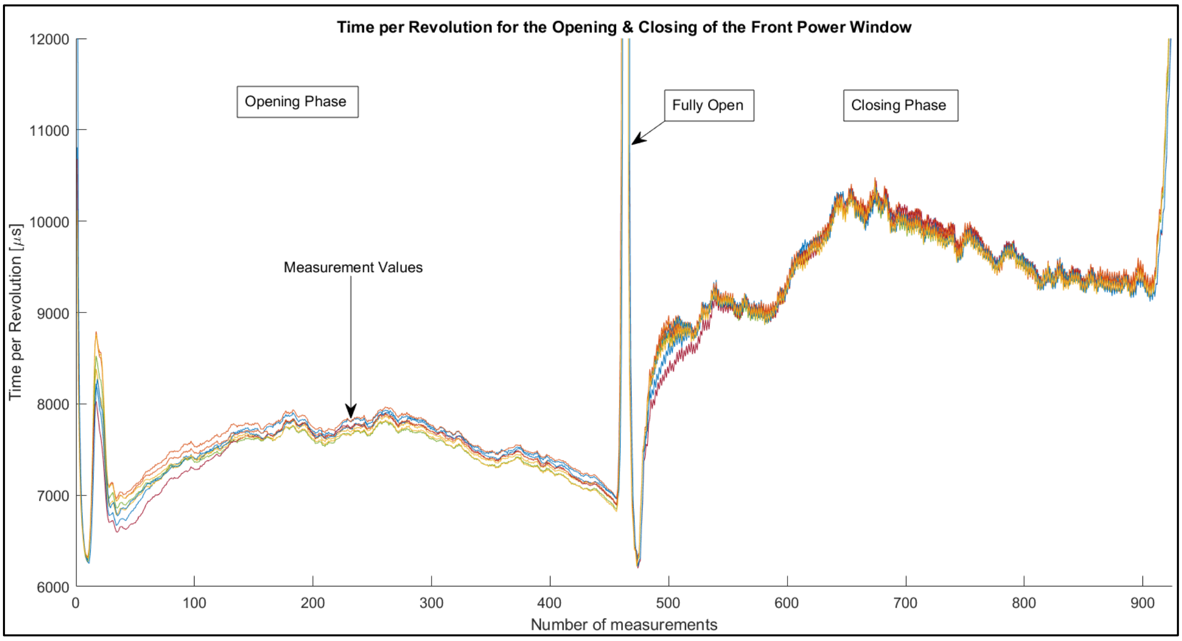


Figure 37 Time per revolution measurements for both the opening and closing of the front power window

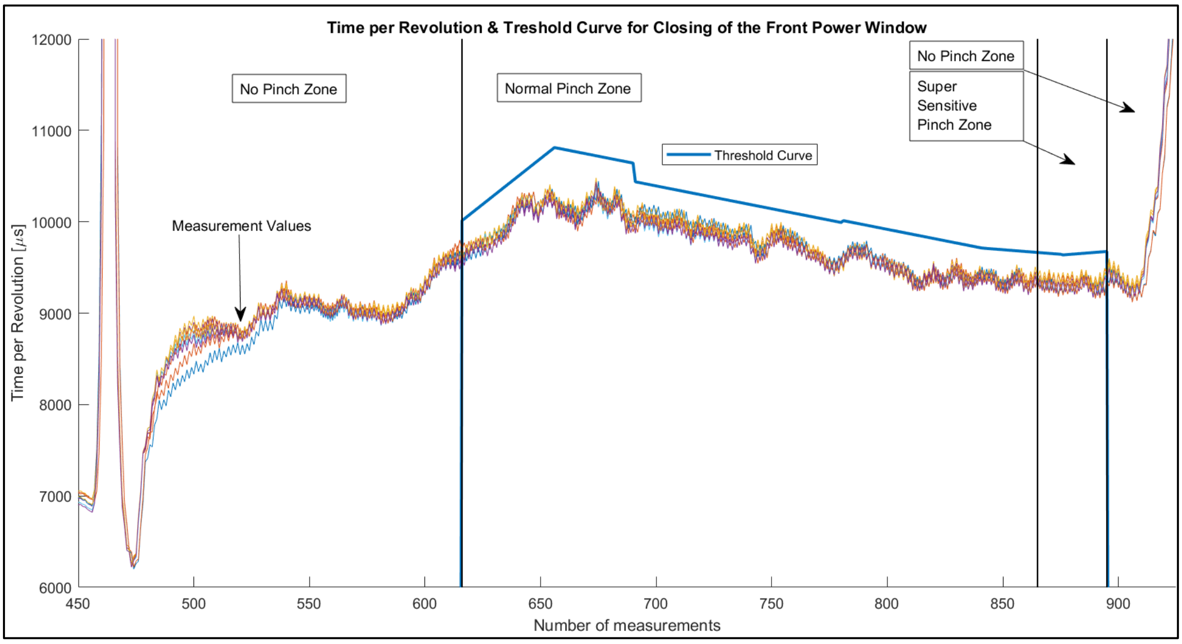


Figure 38 Time per revolution measurements for the closing phase of the front power window, the threshold curve can be seen for the parts where the anti-pinch system is active

7.3 Prototyping Concept 2

A lot of the basic software and hardware developed for concept 1 was reused for concept 2. By doing so, a lot of time was saved, and also made it easy to switch between the two concepts to compare and illustrate their differences.

7.3.1 Prototyping Hardware

The hardware for concept 2 comprised of mostly the same hardware as for concept 1, but instead of using relays for changing the direction of the motor, a motor controller was used. The motor controller is a reference design from Texas Instruments called the DRV8703-Q1EVM, shown in Figure 39. This reference design is based on the DRV8703-Q1 Automotive H-Bridge Gate Driver, consisting of a single H-bridge gate driver that can make use of four external MOSFETs to rotate a single brushed DC motor in two directions at different speeds. Speed and directional control were attained using PWM signals generated from Arduino’s timers 1 and 2.

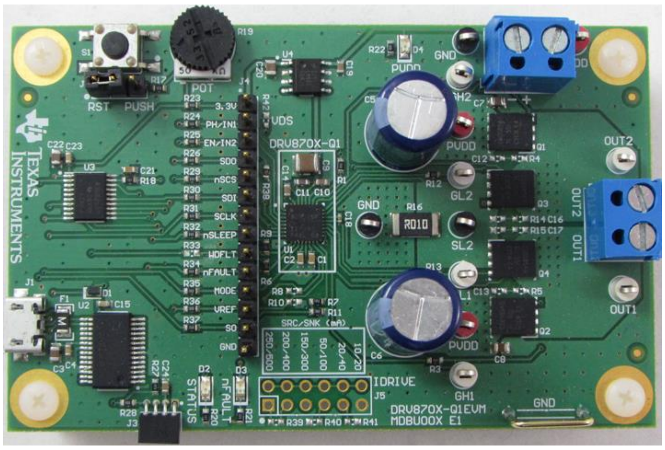


Figure 39 DRV8703-Q1EVM motor controller from Texas Instruments (Texas Instruments, n.d.)

The fluctuation from the PWM signals caused a lot of measurement noise in the hall sensors' output. In order to get a low-noise, reliable signal, an RC low-pass filter was introduced. The resistance and capacitance values used in the RC filters for the panel, curtain and the window are listed in Table 10 together with the cut-off frequency of the filters. The circuit used for concept 2 is illustrated in a circuit diagram in Figure 41, and an image showing the actual circuit can be seen in Figure 40.

Table 10 The table shows the resistance and capacitance values together with the cut-off frequency for the RC-filters in the panel, curtain and window

Component	Resistance (kΩ)	Capacitance (nF)	Cutoff Frequency (kHz)
Panel	4	4.7	8.47
Curtain	200	0.15	5.31
Window	510	1	0.31

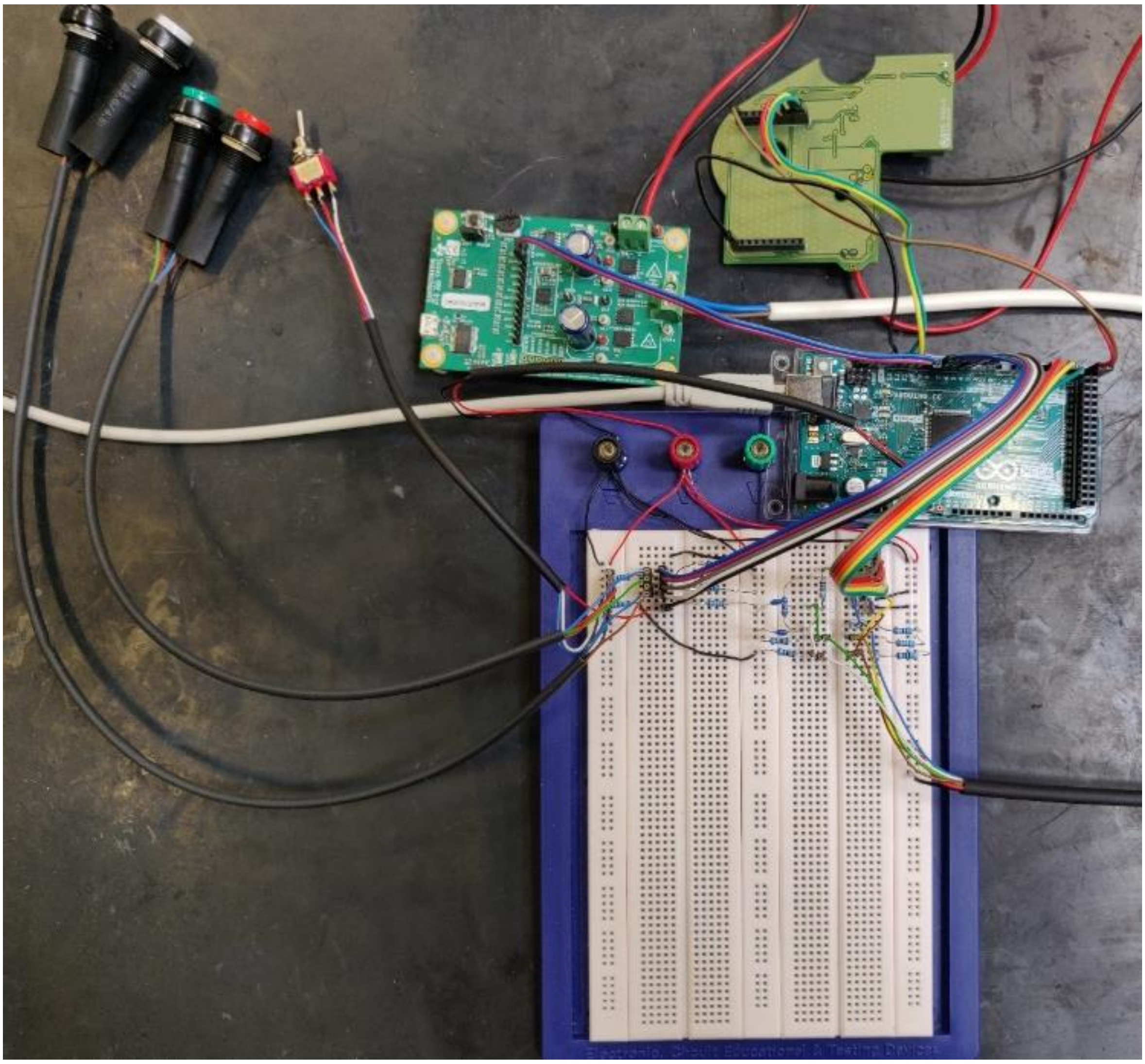


Figure 40 The image shows the circuit used for concept 1 and 2. The image includes the Arduino, two motor controllers, the wiring from the hall sensors, the buttons and the RC-filters

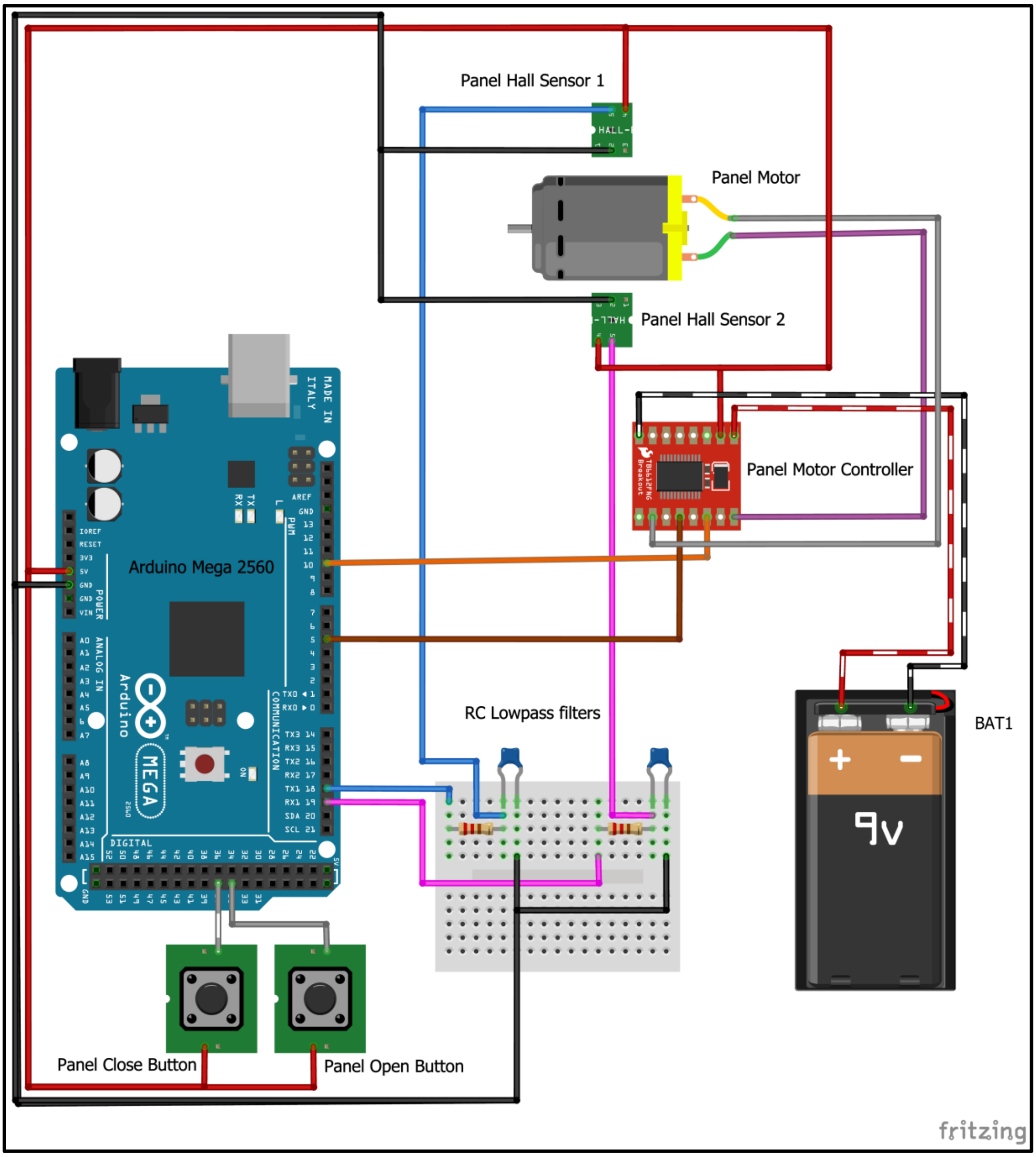


Figure 41 Circuit diagram for concept 2 panel

7.3.2 Prototyping process Concept 2

Similar to concept 1, the prototyping was started with the sunroof panel. When the panel could be operated reliably, the hardware reused and the software were copied with some modifications to make them work for the curtain and then, finally the power window.

7.3.2.1 Sunroof Panel Concept 2

The first challenge with building the circuit for concept 2 was to replace the relays in concept 1, with the motor controller from Texas Instrument. In order to do that, a PWM signal of 31.272 kHz frequency was generated from the Arduino Mega. A high frequency was required by the motor driver to reduce noise and increase the reliability of the operation. The motor controller was used for controlling both the duty cycle and direction of the current, meaning that it could change both the speed and direction of the panel.

As described in Sunroof Panel Concept 1, the movement of the panel is a bit complex. For most parts of the range of motion it is sliding in the horizontal plane, but it also has a tilting motion as shown in Figure 28. For the region where the sliding occurs, the speed could be reduced drastically, but in the region where the tilting motion was performed, the speed could only be reduced with a few percentages. If it was reduced by a bigger extent, the panel stops moving and gets stuck in the tilting region. The speed was thereby reduced to different extents in different zones.

But, changing the PWM duty cycle brought with it more difficulties than just the panel getting stuck at lower speeds. Whenever the panel was driven by a duty cycle lower than 100%, there was a huge increase in measurement noise from the hall sensors. Over the full range of motion, more than 30 000 hall sensor inputs could arrive, which can be compared to 1385 for concept 1. To minimize the measurement noise a lowpass RC filter was built. After a few iterations with different resistance and capacitance values, a filter with a cut-off frequency of 8.47 kHz was used. Even though the RC filter was implemented, an additional 30 hall sensor inputs were received over the entire range of motion, giving a fully open distance of 1425. These few false hall sensor inputs are registered by the Arduino like if two signals were coming at the same time, giving a time difference of 0 between these two inputs. As the system made use of four hall sensor inputs when calculating the time per revolution, these false inputs generated steep valleys of half the magnitude in the plots of the time per revolution. Since these false inputs never affected the performance of the system, they were accepted without any countermeasures.

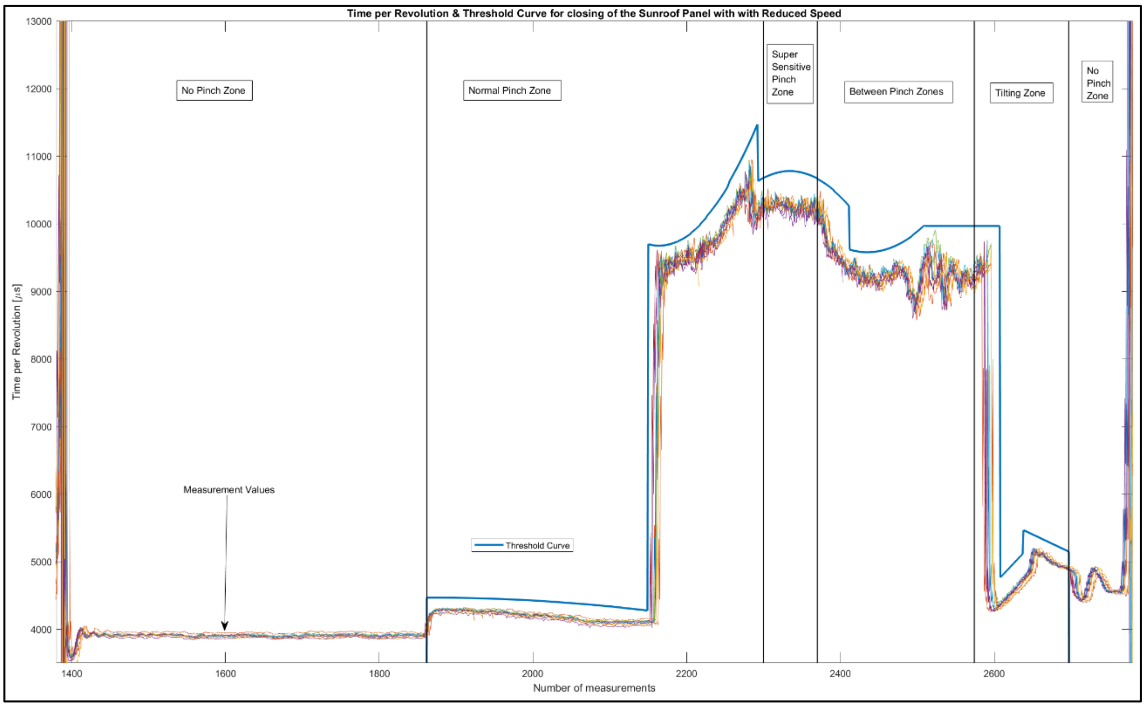


Figure 42 Time per revolution measurements for the closing phase of the sunroof panel, the threshold curve can also be seen for the parts where the anti-pinch system is active. The increased amplitude of both the measurements and the threshold curve indicates a lowered speed in that region.

Just as in concept 1, ten consecutive test runs were made to gather measurements of how the time per revolution varied. Following this, a threshold curve based on these values was created using the same method as in concept 1. The new threshold curve and the time per revolution measurements are shown in Figure 42. The reduction in speed appears significantly in the plot.

During the prototyping of concept 2, a very inconsistent and nonlinear relationship between the reduction of PWM duty cycle and the increase in time per revolution was noticed. This can be seen in all plots of the time per revolution for the panel, curtain and power window. For the panel, the PWM duty cycle in the “No Pinch Zone” is 100%, when it enters the “Normal Pinch Zone” the duty cycle is set to 91% resulting in an increase in the time per revolution with around 300µs. In the “Super Sensitive Pinch Zone” the PWM duty cycle is set to 88% but here the increase in time per revolution is approximately 6400µs.

An additional troublesome behavior in the time per revolution values appeared when the duty cycle was less than 100%. The variation in the time per revolution values increased drastically with a lower duty cycle. In Figure 43, the variance at every point of the ten time per revolution measurements from concept 1 and concept 2 is plotted. The figure shows significant increase in variation in the regions where the PWM duty cycle is less than 100%. In the “No Pinch Zone” with a duty cycle of 100%, the variance is around 80 between the highest and lowest time per revolution values at a point. This stays relatively unchanged in the beginning of the “Normal Pinch Zone” with a duty cycle of 91%. After that, when the duty cycle is set to 88%, the variance reaches around 300 with a lot of peaks that go up to 1000. The difference in variation can also be seen in Figure 42 where the measurement values in the “No Pinch Zone” are closer to each other compared to the regions with a reduced speed. In the regions with a duty cycle of 100%, the variance in concept 2 was observed to be lower than that in concept 1. This can be explained by the better current regulation of the motor controller compared to the relays used in concept 1.

The purpose of reducing the speed of the panel is to get a more sensitive system with a lower pinch force, but the increased variation in the curve counteracts this. Due to the increased variation in the measurements, the threshold curve needs to be higher above the time per revolution curve to avoid false pinches, reducing the sensitivity of the system.

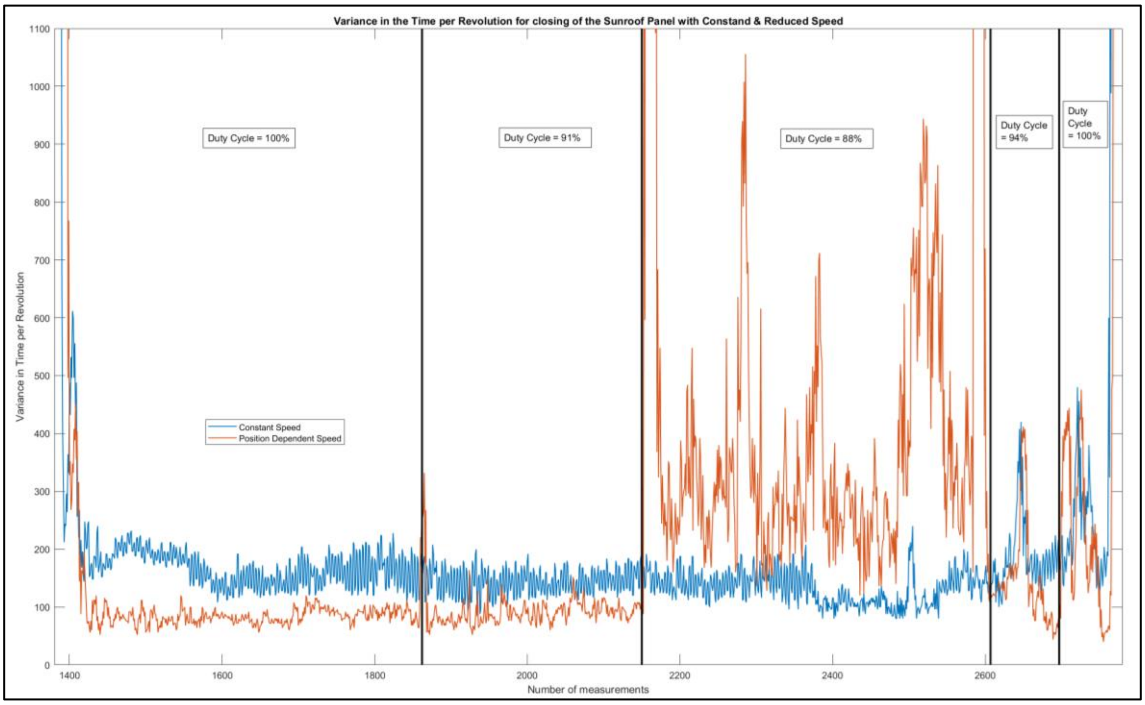


Figure 43 The plot shows the variance in the ten time per revolution measurements runs for both concept 1 with constant speed and for concept 2 with position dependent speed and how it varies with the changes in the PWM duty cycle

7.3.2.2 Sunroof Curtain Concept 2

Reducing the speed for the sunroof curtain was less challenging than for the panel. Mainly because the curtain only performs a sliding motion, thereby avoiding the difficulties that arose from the tilting movement of the panel. The speed of the curtain could be lowered to a great extent without it getting stuck. The other side-effects experienced for the panel, like a noisier signal from the hall sensors, false hall sensor inputs and measurement variation were much less evident in the curtain. After implementing a lowpass RC-filter with a cut-off frequency of 5.31 kHz, no false hall sensor inputs were received. The measurement variance was smaller for the entire range of motion compared to the curtain in concept 1, except in the region with reduced speed, where the variance was similar for both concepts.

The speed was only reduced in the last 50 mm before the curtain reached its fully closed position. In this last region, the PWM duty cycle was set to 78% causing the time per revolution to increase to around 6200 µs compared to 4200 µs for the other regions. Again, ten consecutive test runs were made to gather data of how the time per revolution changes over the range of motion. The threshold curve was created based on the time per revolution measurements using the same iterative method as used for concept 1. The measurements and the threshold curve can be seen in Figure 44.

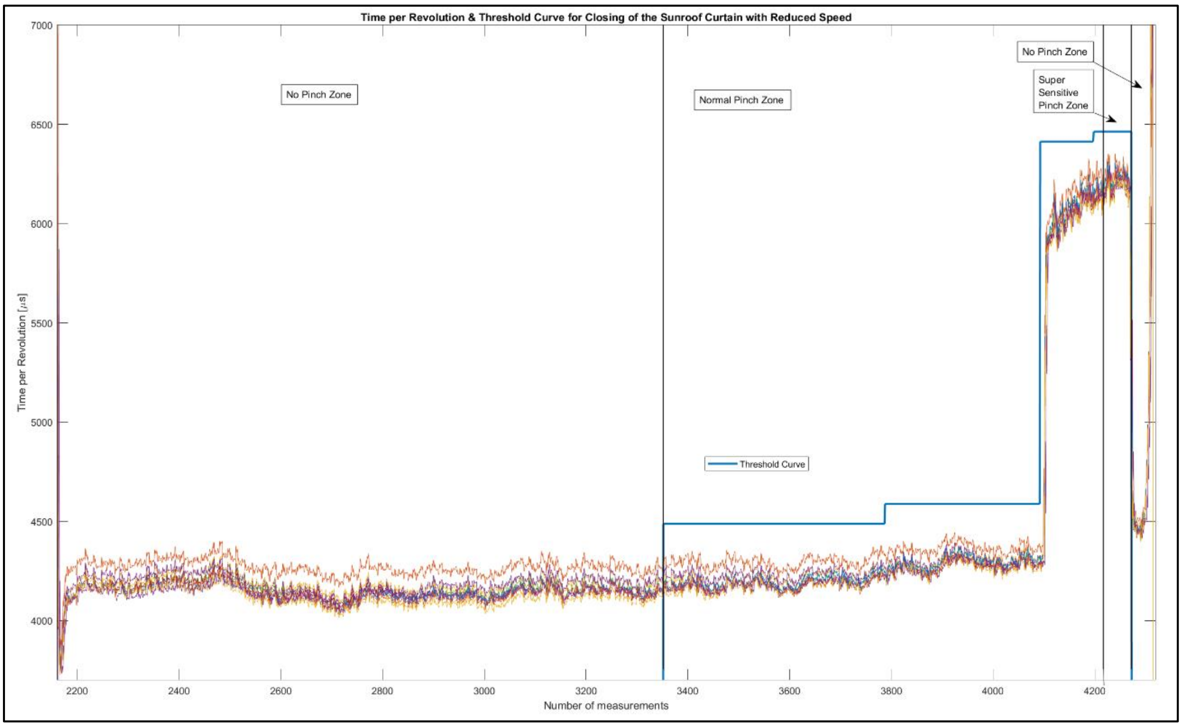


Figure 44 Time per revolution measurements for the closing phase of the sunroof curtain, the threshold curve can also be seen for the parts where the anti-pinch system is active. The increased amplitude of both the measurements and the threshold curve indicates a lower speed in that region

7.3.2.3 Power Window Concept 2

Implementing the speed reduction in the window was similar to the curtain. The base software was already created and the only things that were done additionally was to build a new lowpass RC-filter and to decide in which regions the speed should be lowered. For the power window in concept 2, only one hall sensor could be used because the other hall sensor was damaged, as explained in the previous subsections. The speed of the window was reduced to different extents in the entire pinch zone, with a smaller speed reduction in the beginning and then increased stepwise in the more sensitive regions. The steps where the speed had been reduced stand out clearly in both the time per revolution measurements and the threshold curve plotted in Figure 45. The variance in the measurement values also increased drastically if the PWM duty cycle was less than 100%, similar to the behavior observed for the curtain. The duty cycle is reduced in the following steps when the window enters the "Normal Pinch Zone": 98%, 94%, 90%, 82%, 58% and increasing to 70% before accelerating back to 100% when it has exited the pinch zone. The variation in the measurements increases with the reduction in duty cycle, which can also be seen in Figure 46, where the variation in the measurements is plotted for the window in concepts 1 and 2. The variation was found to be the biggest in the "Super Sensitive Pinch Zone" which can also be seen in Figure 45. This forces the threshold curve to be placed high above the time per revolution measurement values in this vital region.

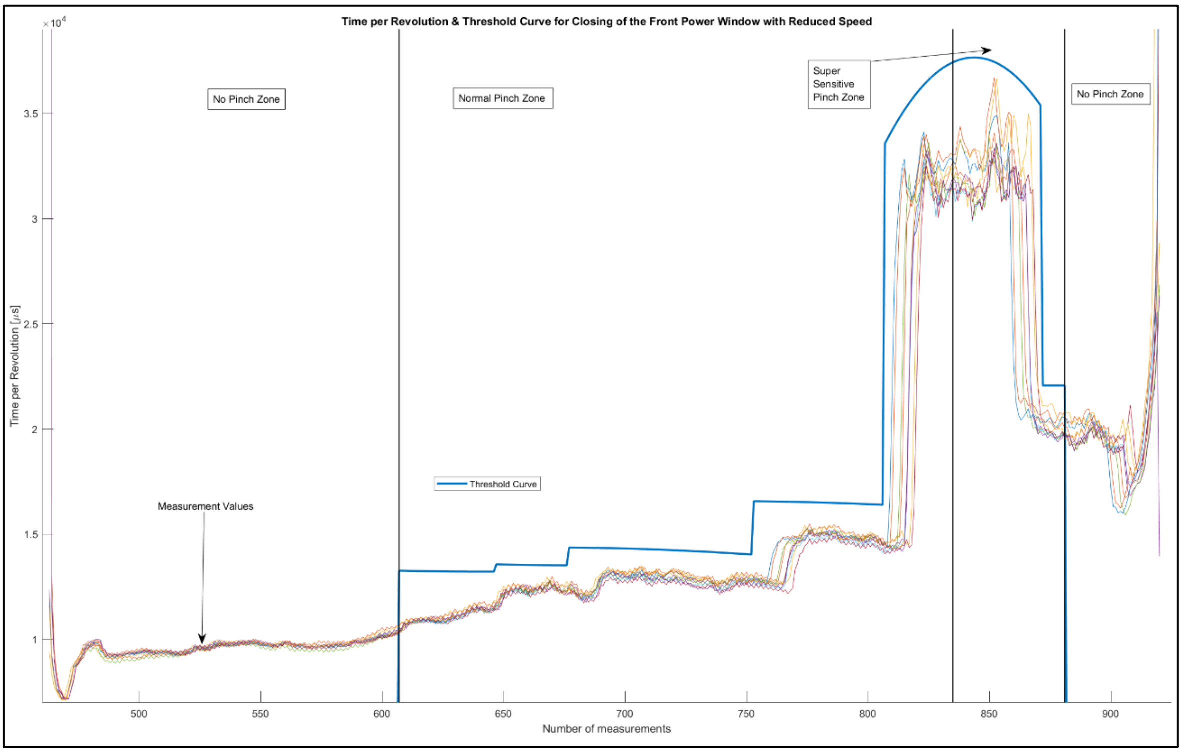


Figure 45 Time per revolution measurements for the closing phase of the front power window, the threshold curve can also be seen for the parts where the anti-pinch system is active. The increased amplitude of both the measurements and the threshold curve indicates a lower speed in that region

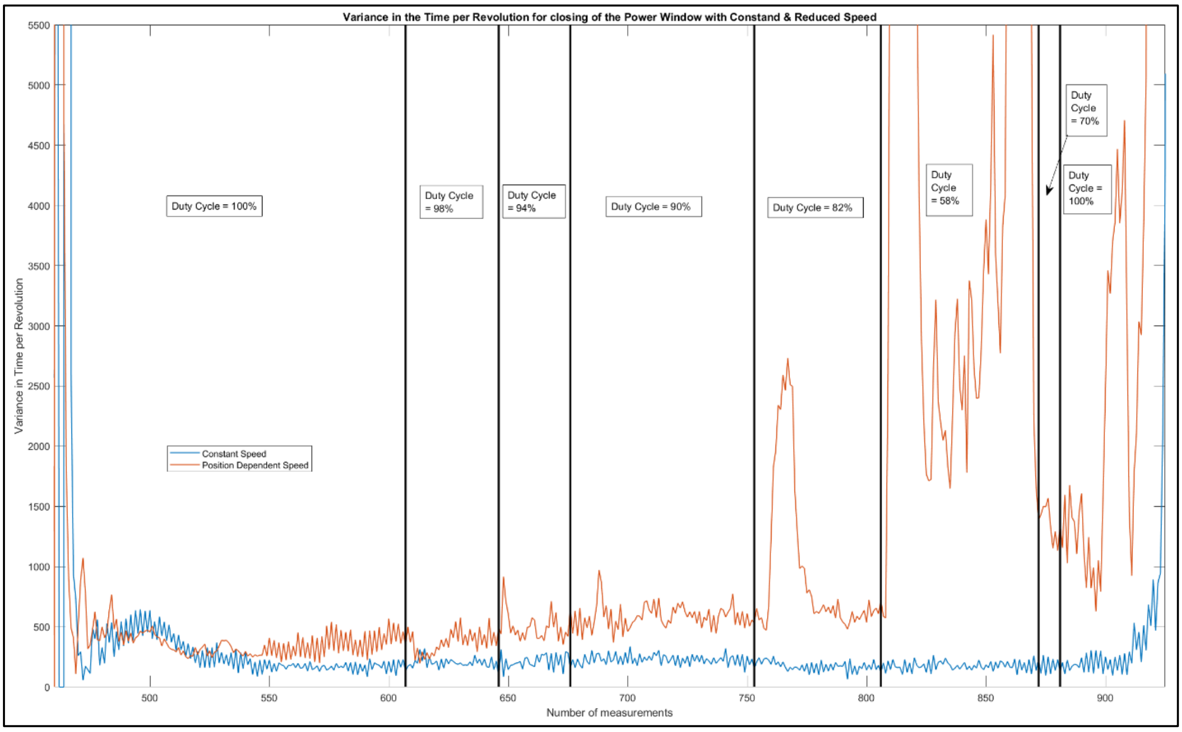


Figure 46 The plot shows the variance in the ten time per revolution measurements runs for both concept 1 with constant speed and for concept 2 with position dependent speed and how it varies with the changes in the PWM duty cycle

7.4 Prototyping Concept 3 and 4

A lot of the basic software and hardware developed for concept 1 and 2 was reused for concept 3 and 4. By doing so, a lot of time was saved, and also made it easy to switch between the concepts to compare and illustrate their differences. This could be done even if concepts 3 and 4 relied on fundamentally different technologies compared to concepts 1 and 2.

7.4.1 Prototyping Hardware

The ripple counting circuit that is the fundamental technology that these concepts rely on is called TIDA 01421. It is an automotive brushed-motor ripple counter reference design for sensorless position measurement from Texas Instruments. This design uses a current sensor, a band-pass filter, a differential amplifier, and a comparator to condition the ripple signals from the motor into 5V signals that can be measured by a microcontroller without requiring additional digital signal processing. An image of the ripple counting circuit can be seen in Figure 47.

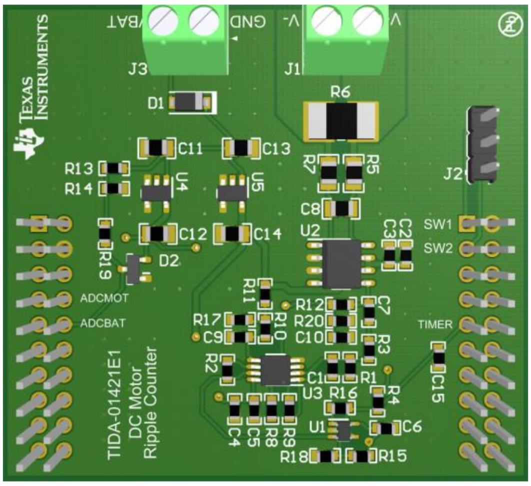


Figure 47 Ripple counter board from Texas Instrument (Texas Instruments, 2018)

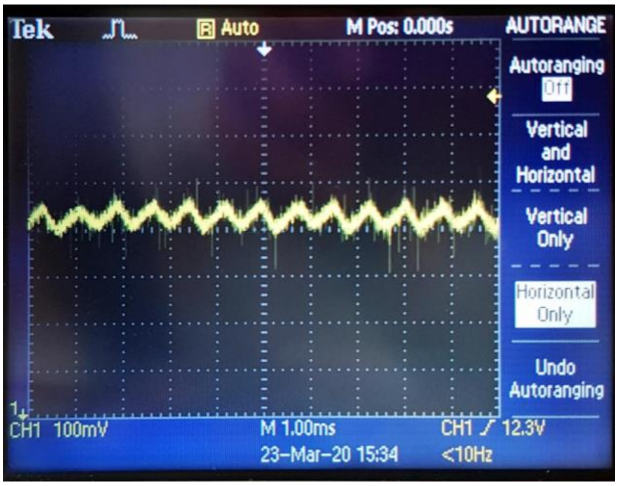


Figure 48 Oscilloscope image of the signal received at the shunt resistor

In theory, ripple counting should have higher output resolution than hall sensor signals. This is because in one rotation of the motor, the number of times the windings short with the motor brushes is much higher than the number of times the poles cross the hall sensor. Unfortunately, the reference design boards that were obtained from Texas Instruments were either not compatible with the tested motors or were faulty. As a result, a usable output signal was not obtained. Figure 48 shows the signals output of the shunt resistor in the circuit.

7.4.2 Prototyping process concept 3

The prototyping process was expected to be similar to that of concepts 1 and 2. The biggest difference would lie in the magnitude of number of position readings for a given range of motion. This would present the need to change ‘expected’ values of various distance related parameters. Apart from this, the other change from previous concept would be the introduction of a Kalman filter in concept 4.

7.5 Software Development

The following chapter describes how the software for the developed concept works. Code snippets from principal parts of the program are provided along with short descriptions of what they do.

7.5.1 Flowchart

The logic behind the software code can be understood by following a simplified flowchart of the program. Figure 49 shows the flow chart for the opening of the panel and Figure 50 shows the closing of the panel.

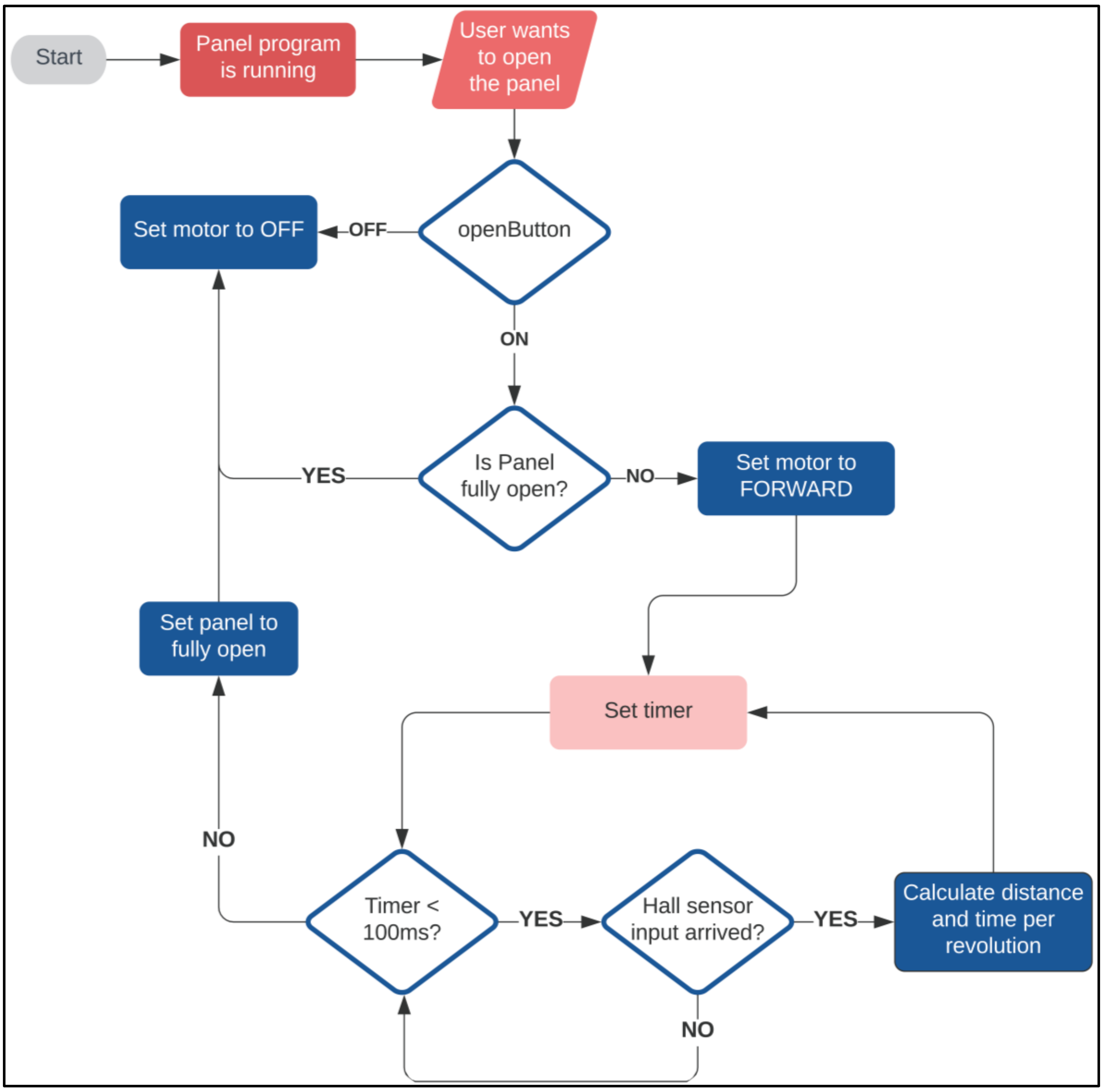


Figure 49 Flowchart showing the opening functionality

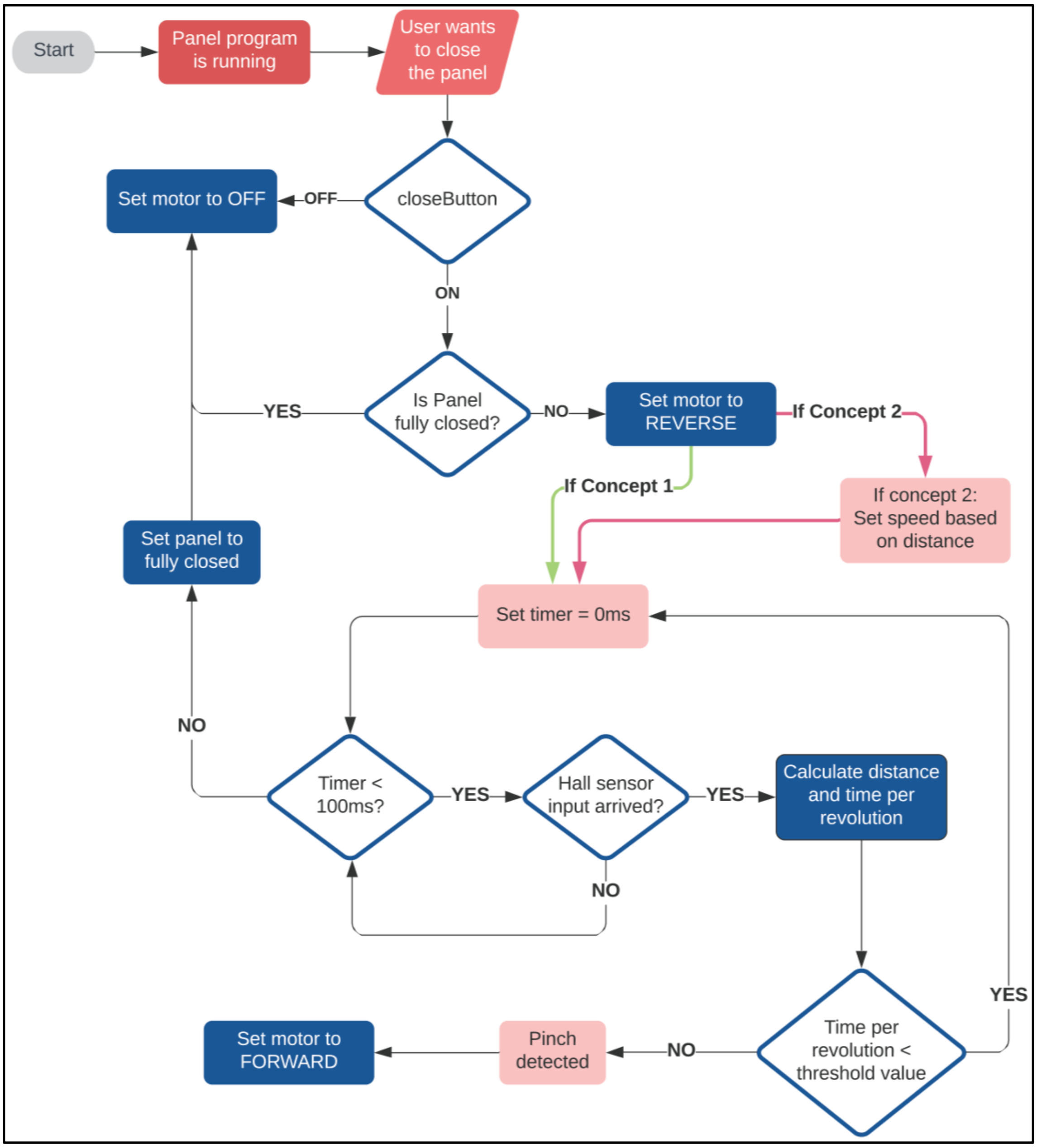


Figure 50 Flowchart showing the closing functionality

7.5.2 Important sections from software code

As mentioned in previous chapters, the MCU used for prototype development was an Arduino Mega 2560. The following sub-sections provide software code snippets and description of how it interfaces with the hardware. The snippets are taken from the software that controls the sunroof panel, but it looks the same for the curtain and the windows.

7.5.2.1 Detecting Hall sensor inputs

One of the most fundamental function of the program is to keep track of the motor's position and speed. This is done by using signals from the motor's hall sensors. When a hall sensor signal arrives, it triggers an interrupt service routine (ISR) and calls the `magnet_detect` function as can be seen in the code snippet below. Interrupts have the advantage of running at a high priority. So, if the program is busy in executing a different task, when the interrupt arrives, the normal program is stopped and the ISR is executed. When the ISR is executed, the normal program can resume. Using interrupts for acquiring hall sensor signals makes sure that the program can always keep correct track of the motor's position and speed.

```
attachInterrupt(digitalPinToInterrupt(hs1Panel), magnet_detect1Panel, RISING); // Interr
upts the system if a hall sensor signal is detected
attachInterrupt(digitalPinToInterrupt(hs2Panel), magnet_detect2Panel, RISING); // Interr
upts the system if a hall sensor signal is detected
```

7.5.2.2 Distance

Each hall sensor signal corresponds to rotational movement in the motor, which in succession corresponds to the movement of the panel. When the panel is opened, every subsequent hall sensor signal will correspond to an increase in the distance by which the panel is opened. Thus, the total number of hall sensors signals received by the Arduino, from when the panel was fully closed to when the panel was fully open, has been used to calculate the total distance the panel travels. As described above, the magnet detect function gets called every time a signal is received from a hall sensor. The `magnet_detect` function consist of an if statement seen in the code snippet bellow. If any button is pressed or a pinch is detected, and the panel is not in its fully open or fully closed position will the `hsDetectPanel` and the `confirmerPanel` be increased by 1. The time at which the hall sensor input was received is also saved in `hs1TimePanel` which is set to the program time.

```
void magnet_detect1Panel()
{
  if ((digitalRead(openButtonPanel) == HIGH || digitalRead(closeButtonPanel) == HIGH ||
pinchDetectorPanel == 1) && fullyOpenPanel == 0 && fullyClosedPanel == 0)
  {
    hsDetectPanel++;
    confirmerPanel++;
    hs1TimePanel = micros();
  }
}
```

After the `hsDetectPanel` is increased by 1 in the `magnet_detect` function, `distancePanel` is increased with 1 by adding the value of the `confirmerPanel` to the `distancePanel`. After that, `confirmerPanel` is set to 0 and the `PaceFunkPanel` function is called to calculate the time per revolution. The `hSensitivityPanel` is equal to 1 and the `hsDetectPanel` will be reset to 0, so this statement is only fulfilled when an input from a hall sensor has been received.

```
if (hsDetectPanel >= hSensitivityPanel)         // Enters when a hall sensor is detected
  {
    distancePanel = distancePanel + confirmerPanel; // Increases the distance
    confirmerPanel = 0;                             // Resets the confirmer
    PaceFunkPanel();    // Calls the pace function to calculate the time per revolution
  }
```

Similarly, when the panel is being closed, the distance value is decreased with every hall sensor signal by subtracting the `confirmerPanel` from the `distancePanel`.

```
if (hsDetectPanel >= hSensitivityPanel)         // Enters when a hall sensor is detected
  {
    distancePanel = distancePanel - confirmerPanel; // Decreases the distance
    confirmerPanel = 0;                             // Resets the confirmer
    PaceFunkPanel();     // Calls the pace function to calculate the time per revolution
  }
```

7.5.2.3 Time per revolution

The PaceFunkPanel function is called every time the distance has been increased or decreased. As described above, the time at which a hall sensor signal arrives is recorded in hs1TimePanel, and the same is done in hs2TimePanel when an input is received from the other hall sensor. The time difference between the sensor inputs is saved in Time_Between_Hs_Input_Panel and then by using the time between four consecutive signals, time per revolution is determined. It is inversely proportional to the rotational speed of the motor, such that a higher time per revolution value corresponds to a slower motor speed, and vice versa.

```
void PaceFunkPanel()
{
 Time_Between_Hs_Input_Panel = hs1TimePanel - hs2TimePanel;      // Inverse of RPM
 Time_Between_Hs_Input_Panel = abs(Time_Between_Hs_Input_Panel); // Take absolute value
 Time_Per_Revolution_Panel=Time_Between_Hs_Input_Panel- Time_Between_Hs_Input_Panel_Old;
 // Checks the difference between the old and new time difference between hs input
  Time_Per_Revolution_Panel = abs(Time_Per_Revolution_Panel); // Take the absolute value
  hsDetectPanel = 0;                                         // Resets Hs detect to 0
  Time_Between_Hs_Input_Panel_Old  = Time_Between_Hs_Input_Panel;   // Updated the Time_
Between_Hs_Input_Panel_Old
}
```

7.5.2.4 Pinch detection

Time per revolution values are plotted against distance values and quadratic curve fits are generated. The pinch zone was split into 7 segments, with each segment having its own curve fit. The equation from these curve fits are then implemented as an inequality statement in the program. When the actual time per revolution value is greater than the expected threshold value from the curve fit, it is defined as a pinch situation. The code snippet below shows the statement that is fulfilled if the distancePanel is within the lower limit, LL1Panel, and the upper limit, UL1Panel, of the first of these 7 segments. If the value of the curve fit for this exact distance is lower than the time per revolution, the pinchDetectorPanel is set to 1, meaning that a pinch has been detected.

```
else if (distancePanel >= LL1Panel && distancePanel < UL1Panel)   // Enters Pinch Zone
  {
    // Checks if the treshold curve for Zone 1 is lower than the "time per revolution"
    if (A1Panel * (UL1Panel - distancePanel) * (UL1Panel - distancePanel) + B1Panel *
(UL1Panel - distancePanel) + C1Panel - pinchSensitivityPanel < Time_Per_Revolution_Panel
)
      {
        pinchDetectorPanel = 1;        // If the treshold curve is lower than "time per
revolution" is a pinch detected --> pinchdetector = 1
      }
  }
```

7.5.2.5 Adaptive curve

The speed of the motor does not remain constant and varies between different closing runs. This implies that a static threshold curve risks to either become too sensitive or too little sensitive, both of which are not ideal. In order to accommodate this variance in the motor speed, the vertical intercept values for the curve are calculated based on time per revolution values from just outside the pinch zones. The quadratic curve fits are defined in the form $a^2x + bx + c$ where the vertical intercept value, c, has the biggest impact on the amplitude of the threshold curve. The c values for all the 7 curve fit segments are set based on the Tresh_base_pos and corresponding C#Panel_tresh values. The Tresh_base_pos is equal to the highest time per revolution measurement received within the last four measurements received, just before the panel enters the pinch zone. This is the adaptive part of the curve since it changes every time the panel is closed. C#Panel_tresh values are pre-determined c values needed to create the desired shape of the curve fit. How the Tresh_base_pos is set and the C#Panel values are determined can be seen in the code snippet below.

```
if (distancePanel > UL1Panel && distancePanel <= UL1Panel + 4) // Adaptive threshold cur
ve
  {
    if (Time_Per_Revolution_Panel > Tresh_base_pos)    // If The time per revolution is 
higher than the treshold base will a new treshold value be given
      {
        Tresh_base_pos = Time_Per_Revolution_Panel;   // A new base value is assigned to
 the treshholdcurve that is unique for this operation
      }
      // Asigned the Adaptive treshold base value to the C values for the curve
      C1Panel = Tresh_base_pos + C1Panel_tresh;           // Zone 1
      C2Panel = Tresh_base_pos + C2Panel_tresh;           // Zone 2
      C3Panel = Tresh_base_pos + C3Panel_tresh;           // Zone 3
      C4Panel = Tresh_base_pos + C4Panel_tresh;           // Zone 4
      C5Panel = Tresh_base_pos + C5Panel_tresh;           // Zone 5
      C6Panel = Tresh_base_pos + C6Panel_tresh;           // Zone 6
      C7Panel = Tresh_base_pos + C7Panel_tresh;           // Zone 7
    }
```

7.5.2.6 PWM generator for concept 2

The motor controller requires PWM signals to set the speed of the motor. The Arduino’s internal timer TCCR1 is used to generate the square wave PWM signal for the curtain and the window. Also, the internal timer TCCR2 is used to generate the square wave PWM signals for the panel.

```
// PWM control changing frequency for TIMER 1
  TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM20);
  TCCR1B = TCCR1B & 0b11111000 | 0x01;

// PWM control changing frequency for TIMER 2
  TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM20);
  TCCR2B = TCCR2B & 0b11111000 | 0x01;
```

The duty cycle and direction of the timers are controlled by registers OCR1A and OCR1B for timer 1, and registers OCR2A and OCR2B for timer 2. The duty cycle can be varied between 0 (no speed) and 255 (full speed). The panel moves in the opening direction when OCR2B is 0 and in the closing direction when it is 255.

```
if (digitalRead(openButtonPanel) == HIGH && fullyOpenPanel == 0)
  {
    fullyClosedPanel = 0;               // Resets the "fullyClosedPanel" flag to 0
    OCR2A = 255;                        // Speed = 100%
    OCR2B = 0;                          // Opening direction
    DistanceOpenFunkPanel();            // Calls the Distance open function
  }
```

```
else if (digitalRead(closeButtonPanel) == HIGH && fullyClosedPanel == 0)
  {
    fullyOpenPanel = 0;                 // Resets the "fullyOpenPanel" flag to 0
    OCR2A = 255;                        // Speed 100%
    OCR2B = 255;                        // Closing direction
    DistanceCloseFunkPanel();           // Cals the Distance closing function
  }                                     // Stops Stop in pinch zone statement
```

8 Concept Testing and Results

The developed prototypes that reached a fully functioning stage were tested so that their performance could be compared to each other and to the requirements placed on them. This chapter is divided into three main parts, first is the testing procedure described, then is the test results showed and finally are the test results analyzed.

8.1 Testing Procedure

The original plan was to get Volvo to perform formal testing of all parts in all concepts against the harsher legal requirements of FMVSS No. 118. Unfortunately, the work limitations due to COVID-19 reduced the available development time and the possibility that Volvo personnel had to conduct the formal testing before the deadline of the thesis. Instead, only the panel and curtain of concept 1 could be tested formally. Since the current supplier cannot fulfill the requirements of FMVSS No. 118, no measurement equipment for testing the sunroof against those requirements was available at Volvo. Instead, tests were performed according to the legal requirements of ECE R21-01. The remaining parts and concepts were tested informally by using a MARK-10 Series 5 force gauge without any spring stiffness specified. The informal testing procedure imitated the formal Volvo testing procedure.

8.1.1 Formal Testing of Panel and Curtain Concept 1

The tests were performed by a test engineer and a mechanic in the strength and endurance laboratory at Volvo Cars in Torslanda. First, a test rig consisting of a metal beam with rails was mounted on the inside of the roof. In one of the rails, an L-shaped beam was mounted and on top of the L-shaped beam a force gauge was placed. The force gauge had a specified spring force of 10 N/mm as required in accordance with ECE R21-01. An L-shaped pressure plate was attached to the force gauge’s measuring rod. With the specified spring stiffness, a force of 100 N would be registered when the spring is compressed by 10 mm. The rails on the rig allowed the position of the force gauge to be easily adjusted in X, Y and Z directions, enabling the gathering of measurements from all specified positions and distances. The test rig set up can be seen in Figure 51.

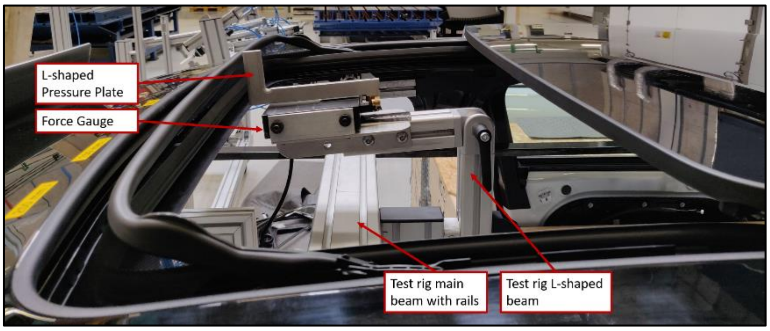


Figure 51 The test rig mounted under the roof, with the force gauge in place. During the test will the panel/curtain move forward until its front edge touches the pressure plate of the force gauge and compresses it to some degree before a pinch is detected. The compression will generate a force that that is noted down and then is the procedure repeated for all measurement points.

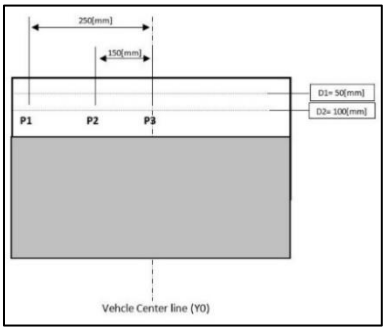


Figure 52 Diagram showing the testing locations for the sunroof panel and curtain. The grey rectangle symbolizes the panel/curtain and the upper black line is the closing edge.

In the ECE R21-01, it is specified that measurements should be taken in three different positions: P1 at the center line of the sunroof; P2 150 mm from the center line; and P3 250 mm from the center line. Since the sunroof is symmetric around the center line, measurements were taken only on one side. Measurements were required to be taken at two distances from the closing edge in all three positions: distance 1 at 50 mm from the edge; and distance 2 at 100 mm from the edge. In each of these six measurement points, five measurements should be taken at each point. For each measurement, the pinch force, the distance of the panel or curtain before it starts to move (distance A), and the distance where the panel or curtain stops after the pinch (distance B) should be noted down. Figure 52 shows a simple diagram of the measurement points. From the five measurement values, the pinch force average and variance at each point was calculated.

Measurements of the pinch force were also taken at the rear edge of the sunroof panel in the tilt zone. Volvo's vehicles have a "tilt function" which gives the user the opportunity to only open the panel to its "tilted" position as showed in Figure 54. When the panel is closing from a tilted position and a pinch is detected, the panel is reversed to the "fully tilted" position. This function does not exist either of the two concepts. As such, if a pinch is detected when the panel is closing from a tilted position, the panel reverses to its fully opened position. This caused problems while using the equipment for measuring the pinch force in the tilt zone. The force gauge was placed on top of the roof, behind the panel with suction cups as seen in Figure 53. After a pinch is detected, the force gauge would be in the way for the panel when it is travelling towards its fully open position. To circumvent this problem, the power to the panel was cut off after each pinch detection, forcing the panel to stop abruptly. This gave time to remove the force gauge and then the system was rebooted. This was cumbersome and time consuming so only one pinch force measurement was taken from each position instead of five as specified in ECE R21-01.

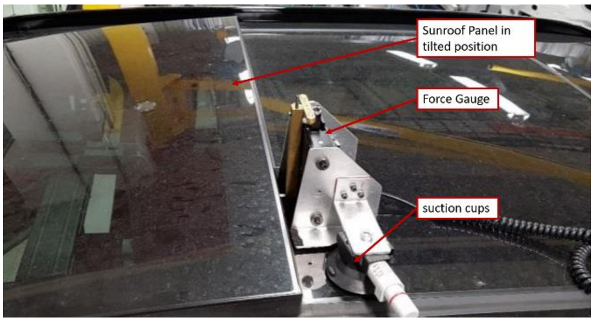


Figure 53 Force gauged placed to measure the pinch in the "tilt zone"

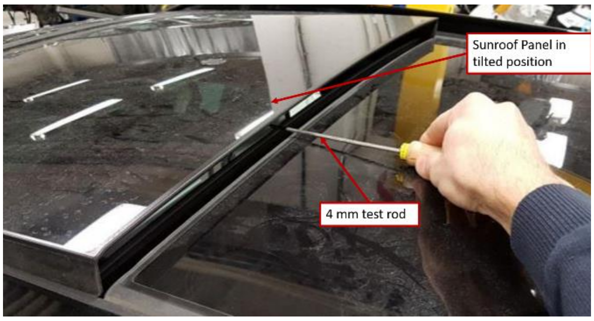


Figure 54 4 mm test rod used to control if the anti-pinch system fulfils demand on minimum pinch distance

The last part of the test was to place a rod of diameter of 4 mm against the closing edge and test if the anti-pinch system registers a pinch at the minimum distance required by ECE R21-01 and FMVSS No. 118. This was done at all three positions on the front edge and in the tilt for the panel, and only at the front edge for the curtain. The test rod used during the 4 mm pinch test at position 1 for the tilt function is shown in Figure 54.

The testing procedure was first performed on the panel as described above. Followed by this, the procedure was carried out for the curtain. Five measurements were taken at distances 1 and 2 at position 3. No measurements were taken from other positions.

8.1.2 Testing of Panel and Curtain Concept 2

Since the formal testing that was done for the sunroof panel and curtain in concept 1 could not be done for concept 2, an informal testing was performed without the standardized equipment. For the informal testing, a MARK-10 Series 5 force gauge was used. The measurement rod in this force gauge is completely stiff unlike the one used in the formal testing which had a spring stiffness of 10 N/mm. A stiff rod should result in higher pinch force values since the distance that the panel moves after coming in contact with the force gauge is minimized, due to the very high spring stiffness of the force gauge.

To gather the measurement values, the force gauge was placed with its bottom against the fully closed edge of the panel. Then the panel or curtain was closed until it pinched against the pressure plate. The set up can be seen in Figure 55. Measurements were taken from the same positions and distances as in the formal testing shown in Figure 52. Additionally, measurements were also taken at a distance of 20 mm, inside the “super sensitive pinch zone”. The pinching distances were changed by changing the length of the measurement rod of the force gauge using the parts enclosed with it.

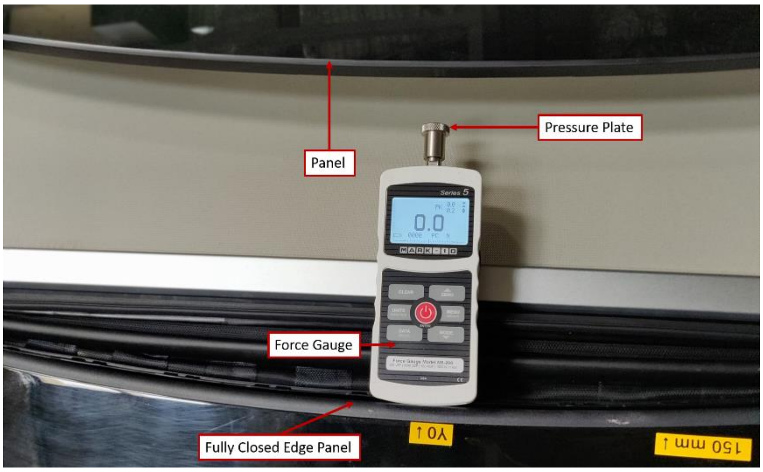


Figure 55 Setup of the force gauge used during the informal testing of the panel. The bottom of the force gauge is placed against the fully closed edge of the panel and the panel is pinching against the pressure plate.

Five measurements were taken at each point; the pinching force and the distance at which the pinch occurred was noted down. The average and the variance of the pinch force values was then calculated. In order to be able to compare concepts 1 and 2, it was required that both concepts were tested using the same method. Therefore, additional measurements were taken for concept 1 using the informal set up. This enabled a fair comparison between the measurement values from concept 1 and 2.

Due to the shape of the force gauge, no measurements could be taken in the “tilt zone”. The testing with the 4 mm rod was conducted in the same way as in the formal testing.

The test set up for the curtain was a little different. When placing the bottom of the force gauge against the fully closed edge of the panel, the height difference was unsuitable for testing, because the curtain could not pinch against the pressure plate. In order to get the readings, a wrench of size 8 was clamped on to the measurement rod of the force gauge creating an L-shape similar to the one used during the formal testing, as shown in Figure 56. With this set up, the force measurements were gathered at all the measurements points on the curtain. The values were gathered from the same positions as were in the formal testing of concept 1. The setup used for gathering the informal measurements for the curtain can be seen in Figure 57.

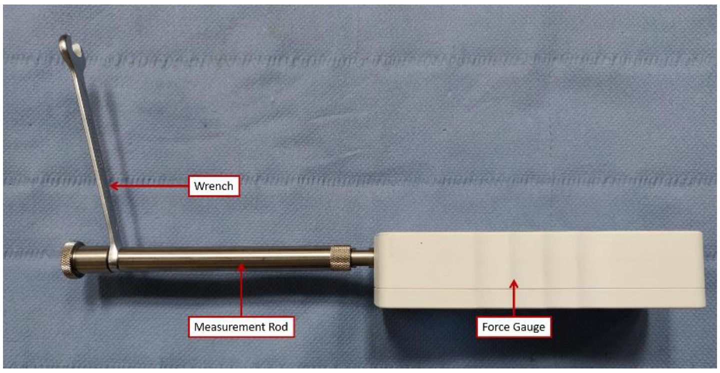


Figure 56 L-shape se up with the force gauge and a wrench

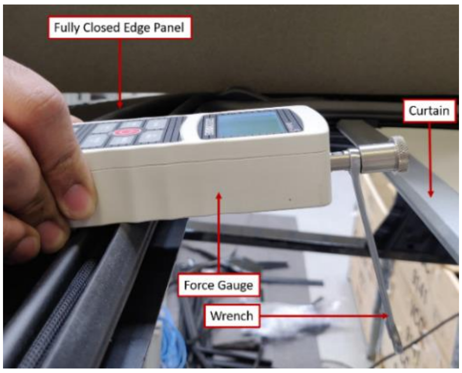


Figure 57 The set up used when gathering the informal measurement from the curtain. The bottom edge of the force gauge is placed against the fully closed edge for the panel and the curtain is pinching against the wrench.

8.1.3 Testing of Windows Concept 1 and 2

The testing procedure for the windows was similar to the testing of the sunroof, but unlike the sunroof the windows are not symmetric. So, the rear edge of the door arc was set as the base from where the measurements were taken at four positions: P1 190mm; P2 500 mm; P3 650 mm; and P4 900 mm. At each position should three distance be tested: D1 25 mm; D2 50 mm; and D3 100 mm from the closing edge. A diagram showing the measurement points at the front door can be seen in Figure 58. At each measurement point, five pinch measurements were recorded and then the average and the variation of the pinch forces were calculated.

The 4 mm test rod was used in the same way as for the sunroof, but for the window the test was conducted at every 50 mm from the rear edge of the door arc over the entire length of the arc.

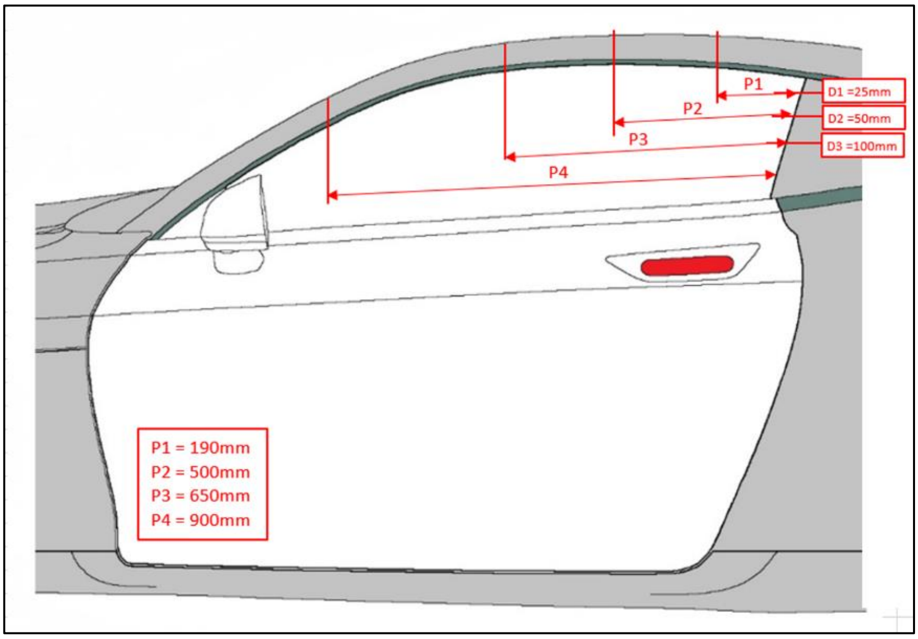


Figure 58 Diagram showing the four positions and three distances for the test procedure of the anti-pinch system in the front window

8.2 Testing Results

A summary of the testing results from the formal and informal testing of the sunroof panel, curtain and windows is presented in following sub-sections.

8.2.1 Results from Formal Testing of the Sunroof Panel and Curtain Concept 1

The complete test results from the testing against the ECE R21-01 legal requirement for the sunroof panel, the tilting of the sunroof panel and the sunroof curtain can be seen in Appendix E. A summary of the maximum, minimum and average pinch forces for the three positions and two distances is listed in Table 11.

Table 11 Summary of the test results for the sunroof panel, tilt, and curtain. The position, distance, max, min and average values are given

Sunroof Panel							
Pos	Distance 1: 50[mm]			Distance 2: 100[mm]			Tilt
	Max Pinch Force [N]	Min Pinch Force [N]	Average Pinch Force [N]	Max Pinch Force [N]	Min Pinch Force [N]	Average Pinch Force [N]	Pinch Force [N]
P1	40	27	37	34	33	34	47
P2	37	37	37	53	32	44	52
P3	49	35	43	56	55	55	48
Sunroof Curtain							
	Distance 1: 50[mm]			Distance 2: 100[mm]			
P3	34	32	33	35	33	34	

8.2.2 Results from Testing of Panel and Curtain Concept 2

The results from the informal testing of concept 1 and 2 can be seen in Appendix E. Table 12 shows a summary of the measurement values from the panel, the maximum, minimum and average pinch force are given for three positions and three distances for both concept 1 and 2. The results from the measurement points are placed next to each other in order to ease the comparison. The same information is given for the curtain in Table 13.

Table 12 Summary of the test results from the informal testing of the sunroof panel. The position, distance, max, min and average values are given for both concepts. The columns showing results from concept 1 have a blue background and those showing results from concept 2 have a green background in order to ease the comparison of the informal results

Sunroof Panel																		
Pos	Distance 1: 20 [mm] Concept 2/Concept 1 [C2/C1]						Distance 2: 50 [mm] Concept 2/Concept 1 [C2/C1]						Distance 3: 100 [mm] Concept 2/Concept 1 [C2/C1]					
	Pinch Force [N]						Pinch Force [N]						Pinch Force [N]					
	Max		Min		Average		Max		Min		Average		Max		Min		Average	
	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1
P1	77	88	38	63	54	74	39	77	27	40	31	53	72	77	52	45	64	62
P2	47	70	38	60	44	64	38	66	32	42	36	52	78	92	52	57	63	76
P3	55	69	28	60	42	66	43	60	18	52	29	56	68	81	44	46	58	62

Table 13 Summary of the test results from the informal testing of the sunroof curtain. The position, distance, max, min and average values are given for both concepts. The columns showing results from concept 1 have a blue background and those showing results from concept 2 have a green background in order to ease the comparison of the informal results

Sunroof Curtain																		
Pos	Distance 1: 25 [mm] Concept 2/Concept 1 [C2/C1]						Distance 2: 50 [mm] Concept 2/Concept 1 [C2/C1]						Distance 3: 100 [mm] Concept 2/Concept 1 [C2/C1]					
	Pinch Force [N]						Pinch Force [N]						Pinch Force [N]					
	Max		Min		Average		Max		Min		Average		Max		Min		Average	
	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1
P1	30	42	21	41	26	42	36	39	26	32	30	35	39	42	33	36	37	40
P2	32	37	26	25	28	30	39	44	30	39	35	41	45	46	34	39	40	43
P3	29	37	23	31	27	34	39	46	30	35	34	41	38	39	35	32	36	37

8.2.3 Results from Testing of Windows Concept 1 and 2

Measurements could not be gathered from all measurement points for the window. Table 14 shows a summary of the received values, the maximum, minimum and average pinch forces are shown in the same way as for the panel and the curtain. If the measurement values are missing for a distance/position, there is a "-" in the box.

Table 14 Summary of the test results from the informal testing of the front window. The position, distance, max, min and average values are given for both concepts. The columns showing results from concept 1 have a blue background and those showing results from concept 2 have a green background in order to ease the comparison of the informal results

Front Window																		
Pos	Distance 1: 25 [mm] Concept 2/Concept 1 [C2/C1]						Distance 2: 50 [mm] Concept 2/Concept 1 [C2/C1]						Distance 3: 100 [mm] Concept 2/Concept 1 [C2/C1]					
	Pinch Force [N]						Pinch Force [N]						Pinch Force [N]					
	Max		Min		Average		Max		Min		Average		Max		Min		Average	
	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1
P1	40	59	31	46	36	53	54	60	40	46	46	53	73	55	60	45	68	52
P2	28	41	19	32	24	36	-	-	-	-	-	-	-	-	-	-	-	-
P3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

8.3 Test Analysis and Concept Evaluation

The results from the two concepts presented above have been evaluated in the following section. The positions that received the highest and lowest pinch forces have been mentioned. The results have been compared to the legal requirements and in-between the different concepts.

8.3.1 Sunroof Concept 1

The pinch forces measured during the formal testing of concept 1 were significantly lower than both the legal requirements specified in ECE R21-01 and Volvo's internal force requirements. The maximum pinch force in the panel was received at position 3, distance 2 and was 56 N. In most regions, the pinch forces were much lower than this, with a minimum pinch force of 27 N received at position 1 distance 1. The three measurements taken in the "tilt zone" gave an average of 49 N which was also below the maximum limit. The number of measurements taken for the curtain were few, but they were taken from the measuring point that gave the highest pinch forces for the panel, positions 3 and for both distances. Thus, the results for the curtain cannot be considered as credible as for the panel due to the lower number of measurements. However, the force values were quite low, around 33 N. So, it can be said that the force requirements were met. To support this, the measurements from the informal test can be used. Despite the informal testing using a completely stiff spring, the maximum pinch force for the curtain was 46 N, which further validates the conclusion that the legal requirements were met for the curtain.

It was unfortunate that only testing against the ECE R21-01 could be performed. In order to overcome this deficiency as much as possible, the informal measurements were the only data points that existed. The FMVSS No. 118 requires that a spring stiffness of 20 N/mm is used for distances between 25 mm and 200 mm from the closing edge and a spring stiffness of 65 N/mm for distances less than 25 mm. The uncertainties in the process of gathering the measurements and the absence of an unbiased third party lowers the credibility of these values. So, the informal testing can only be used as an indicator of how the concept would perform when tested against the FMVSS No. 118. However, with a maximum pinch force of 92 N and an overall average of 63 N, it is very likely that the concept would fulfill the FMVSS No. 118 requirement and the internally set Volvo requirement of 75 N.

The pinch force is directly proportional to the sensitivity of the system. Thus, a system which gives a lower pinch force value would be more sensitive. Unfortunately, a more sensitive system has a higher risk of detecting false pinches. To this end, there is no reason for developing a system that results in a lower pinch force than the one specified by Volvo Cars. From the test results, it can be concluded that Concept 1 was much more sensitive than it needed to be for meeting the ECE R21-01 requirements. The sensitivity seems to be correctly balanced to fulfill the requirements in the more sensitive zone of the FMVSS No. 118 with a spring force of 65 N/mm.

8.3.2 Sunroof Concept 2

For Concept 2 all the testing data come from the informal testing. So, the uncertainties described for the informal testing of Concept 1 apply here as well. The maximum pinch force in the sunroof panel was 78 N at position 2, distance 3. This is at the same location that gave the maximum pinch force of 92 N for the informal testing of Concept 1. The minimum pinch force is 18 N, received at position 1 distance 2, which is 9 N lower than the minimum pinch force received during the formal testing of concept one, but here with a stiff measurement rod. For the panel, the average pinch force values were observed to be generally 15-20 N lower for concept 2 in comparison to concept 1. Which in turn can be related to the values received during the formal testing of concept 1, where it was concluded that the system fulfils the pinch force requirements and was very sensitive. Based on this, a similar conclusion can be drawn for concept 2 which produced even lower pinch force values. And as explained earlier, the variation in the time per revolution values being much higher in concept 2 increases the risk for false pinches.

The same argument applies for the curtain. However, the risk for false pinches caused by external factors like wind or temperature is lower for the curtain since it is places further into the vehicle than the panel. This means that a more sensitive system for the curtain does not impose the same risks of detecting false pinches as in the panel. The pinch forces received during the informal testing for the curtain in concept 2 are extremely low with an overall average of 32.5 N.

8.3.3 Window for concept 1 and 2

Since a stable frame or rig could not be built to affix the doors in their correct position, reliable measurement values could not be gathered. Additional problems existed with the doors, comprising of many different factors. The improvised connections and the dead hall sensors have been explained in earlier chapters. In addition to these, the absence of a frame/rig resulted in that the doors were standing in the EUR-pallets, in which they were received. The position measurements of the window kept changing randomly due to the lack of structural support. This resulted in randomly changing load cases for the motor, and thereby varying time per revolution measurements. Since the windows are moving in the vertical plane, they are very sensitive to changes at which angle the door was standing at. Tilting the door more inwards would increase the contact area between the window and the seal on the inside. This led to increased friction, which consequently changed the load case for the motor causing the shape of the time per revolution curve to alter. These varying load cases resulted in either an unexpected increase of the pinch force values or an unexpected increase in the number of false pinches. To address this, new time per revolution measurements and new threshold curve were needed.

These problems were present during the entire development process. Another issue that arose during the informal testing due to the substandard fixing of the doors, was that when the window pinched against the force gauge, the resulting reaction force caused the door to move some measurements. This resulted in lower and misleading pinch force values.

After gathering the force values from all three distances at position 1 and from distance 1 at position 2 for both concepts, it was decided to stop the testing due to the low quality of the test values. During the process of gathering the test values, a lot of measurements were forced to be retaken, due to extreme low pinch forces resulting from the movement of the door. At one point, the actual speed of the window had strayed away from the threshold curve so much that false pinches were detected before the window glass could reach the force gauge.

Figure 59 shows one of the last futile attempts of fixing the front door in one position using tension straps and an applied force from the side.

Despite the failed testing and the low quality of the gathered pinch forces can it be noted that all measurement values are below both the Volvo and legal requirements. The testing could be redone at a later stage if the concepts are further developed and the doors can be fixed in their correct position. The most important thing for the windows might not have been to gather the exact pinch forces, but to prove that a software developed for the sunroof can be implemented in the windows without any major modification.

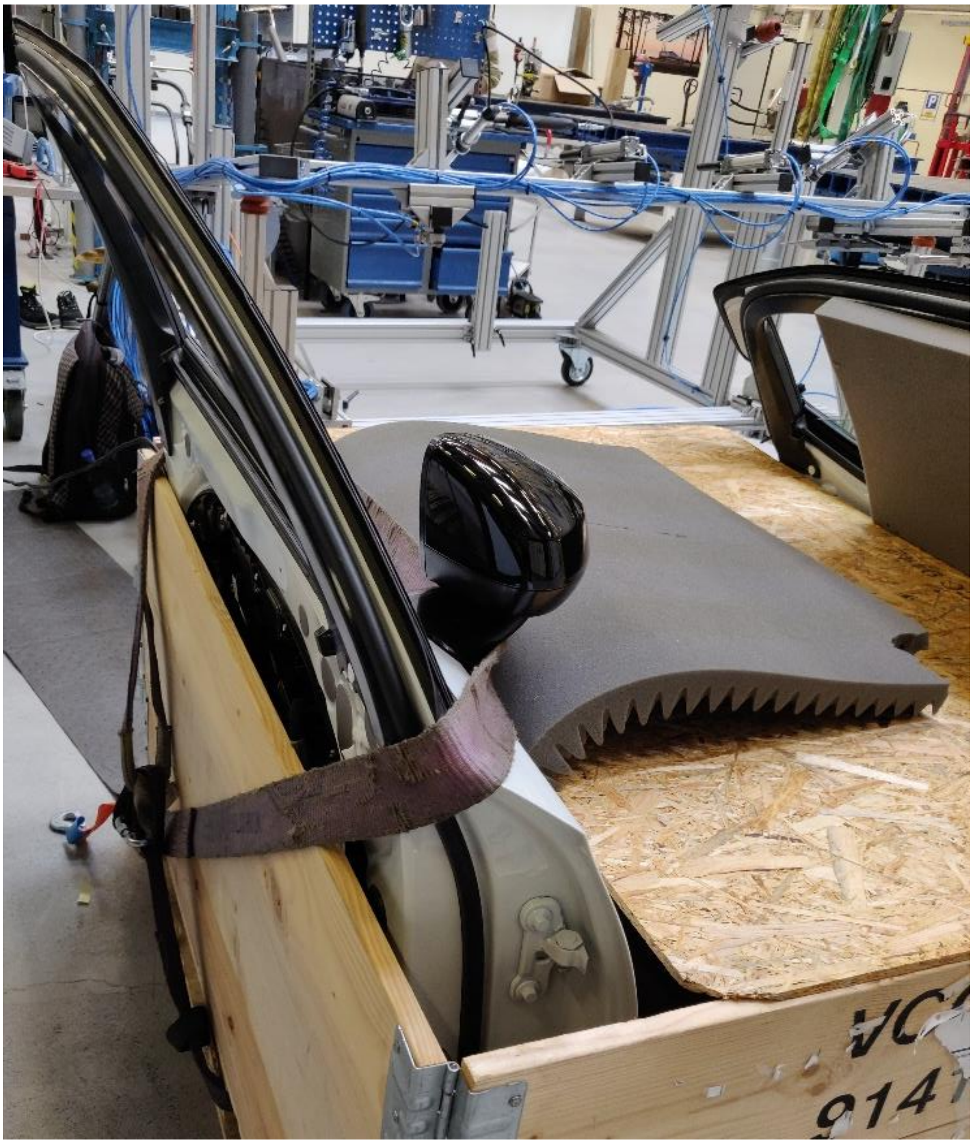


Figure 59 One of the attempts of securing the front door in one position so that reliable measurements could be taken without that the door moved

9 Cost Estimation

This chapter is divided into two parts that detail the estimation of the development costs and the costs for the components needed in the different concepts. The estimations for the development costs have been done for two scenarios, the current scenario where all development and calibration were done by different suppliers, and for a scenario where both the development and the calibration were done by Volvo Cars. In addition, the chapter contains information about the development and component costs for the different concepts.

9.1 Cost Estimation of Outsourcing

Estimating the cost of the system currently used by Volvo was hard, since the anti-pinch and position control systems are a part of the complete sunroof package. This package includes the entire hardware for the panel, curtain, controller, etc., and the anti-pinch system. The situation is similar for the windows.

The following cost calculations were based on data received from the product owners responsible for the anti-pinch system in the sunroof, windows and tailgate at Volvo Cars. The costs for the anti-pinch system in the tailgate is included since it is assumed that that the developed concepts could be implemented there as well. If no exact values could be obtained, they had been estimated by the product owner. These estimates were made conservatively in order to accommodate for uncertainty. The calculations were done for the 6 models on the SPA1 platform and a production volume of 100 000 cars per model. The total cost estimation can be seen in Appendix D.

The D&D cost for outsourcing the anti-pinch system consists of three main parts, the development cost, the calibration cost and the resources required by Volvo for administrating everything related to the anti-pinch system. These costs for the sunroof, windows and tailgate can be seen in Table 15.

Table 15 Cost calculations for outsourcing the anti-pinch system for the sunroof, windows and tailgate

Cost Description [SEK]	Sunroof	Windows	Tailgate
Development	10 000 000	5 000 000	500 000
Calibration	4 295 654	2 562 600	2 780 000
Volvo resources administration SPA1	1 600 000	400 000	400 000
D&D Sunroof Anti-pinch cost SPA1	15 895 654	7 962 600	3 680 000
Cost per vehicle	26.49	13.27	6.13

In addition to the D&D costs listed in the table, there is a license cost that Volvo pays for every vehicle. The license cost is 8 SEK for the sunroof, 32 SEK for the windows and 8 SEK for the tailgate, resulting in a licensing cost of 48 SEK per vehicle and total of 28 800 000 SEK for all vehicles in SPA1, if 100 000 vehicles were manufactures for every model.

$$License\ cost\ per\ vehicle = 45\ \times 6\ \times 100\ 000 =\ \ 28\ 800\ 000\ SEK$$

This gives a total cost per vehicle of 90.89 SEK for the D&D and licensing if the anti-pinch system is outsourced.

9.2 Cost Estimation of In-house Development

Since no anti-pinch or position control systems has been previously developed at Volvo Cars, the following estimations were based on assumptions. The base cost of having one engineer working fulltime with a project for one year is 1 MSEK. Considering the progress that was made in 20 weeks by a project group consisting of two master's students with no previous experience within the area, it was estimated that two experienced engineers with the relevant knowledge, working full time, would be able to develop a system ready for implementation in 6 months. The estimated cost for this would be 1 MSEK. The prototypes presented in an earlier chapter were built at a cost of less than 10 000 SEK. Since the development of an anti-pinch system utilizes vehicle parts from other projects, no expensive prototypes would be needed and thus costs related to prototyping can be neglected.

It is also believed that the calibration costs can be reduced drastically if it was done within the organization. For example, for calibration of some of the parts, the suppliers charge 9500 SEK per hour, which could be done for a much lower cost by a Volvo engineer. It was assumed that one engineer working full time for a year could do the calibration, resulting in an estimated calibration cost of 1 MSEK. The development and calibration cost give a total cost per vehicle of 3.3 SEK for the models on the SPA1 platform. Important to note here is the fact that the total estimated cost of 2 MSEK for developing the system in-house, is lower than the administration cost of 2.4 MSEK that Volvo currently pays for the outsourcing of the anti-pinch system.

According to these estimates, the development of an in-house anti-pinch system would reduce the associated cost per vehicle from 90.89 SEK to 3.3 SEK.

9.3 Estimation of Component Costs

All concepts, including the current systems bought from suppliers, share some common components that will be the same, independent of the solution. These components are called sunroof base components and window base components. The sunroof base components consist of two motors, one each for the panel and for the curtain, the necessary wiring and one control unit placed in the overhead console (OHC). The window base components consist of four motors, one for each window, the wiring and four control boards one in each door, and the door control unit (DCU). The assembly costs for the control boards are also included for both the sunroof and the windows. Since these costs are the same for all concepts, they were not considered when calculating the component costs for the concepts.

The cost calculations for concepts 1, 2, 3 and 4 can be done relatively accurately since all necessary components were bought for building the prototypes. For concepts 6, 12 and 16 the estimation was harder since similar solutions did not exist at Volvo.

When buying electrical components does the price vary a lot depending on the batch size, the price is decreasing drastically when buying more units. In order to estimate the component cost for the concept 1, 2, 3 and 4 was the price per unit checked for buying 1000 units of every component used when building the prototypes. This price was then used to calculate a total cost for building each concept and then reduced by an additional 80 % since Volvo would buy much more than just 1000 components and they could have a special agreement with the supplier that would lower the cost further. The same approach was used to estimate the component cost of the systems currently bought from suppliers. Table 16 shows the price when buying 1000 units of the automotive graded relays, the motor controller and the ripple counting board from Texas Instruments used during the prototyping. The price information comes from the webpage Digikey.se which were used for buying all components for the prototypes.

Table 16 Cost calculations for automotive grades relays, motor controller and ripple counting boards form Texas Instruments. Both when buying 1000 units of each and for mass production

Board / Components	Price / piece if buying 1000 pieces [SEK]	Mass manufactured Price per piece
Relays	75	15
Motor Controller	47.67	13.259
Ripple Counter Board	66.297	9.534

The current solution bought from suppliers looks differently in the sunroof and for the windows. The sunroof doesn't have any speed control which means that relays could be used or a motor controller without changing the PWM duty cycle. The windows have some sort of speed control since they slowdown in the most sensitive region. The cost for automotive graded relays is relatively high, the relays used during the prototyping were not automotive graded so in order to get any price data was an average cost of 75 SEK used for buying 1000 units. This is a very high cost since 2 relays are needed for every motor and the same functionality could be achieved using one motor controller per motor. It is not known if the current solution uses relays or motor controllers or any other solution to change the direction of the sunroof and windows, so both price estimates are given. The cost per vehicle using the current solution would be 113 SEK if using relays and 79.55 SEK if using motor controllers.

Concept 1 only need relays to control the sunroof and the windows, but since it works just as good with one motor controller per motor and since that would be a cheaper option is both cost estimates given. The cost per vehicle using concept 1 would be 180 SEK if using relays and 79.55 SEK if using motor controllers.

Concept 2 must use motor controllers since it changes the speed of the sunroof and the windows. Using one motor controller per motor gives a total cost per vehicle of 79.55 SEK.

Concept 3 can just as the current solution and concept 1 use both relays and motor controllers, in addition to this is a ripple counting board needed for every motor. The cost per vehicle using concept 3 would be 237.2 SEK if using relays and 136.75 SEK if using motor controllers.

Concept 4 also uses speed control and must have motor controllers, and just like concept 3 is a ripple counting board needed per motor. Using one motor controller and one ripple counting board per motor gives a total cost per vehicle of 136.75 SEK.

For estimating the component cost for concept 6, 12 and 16 is the process a bit vaguer sine these components have not been used during the prototyping process in this project. However, all three concepts must have some form of relays or motor controller for the position control of the sunroof and windows. This gives them a minimum cost equal to the cost of using 6 motor controllers, then should the cost for the components needed to detect a pinch be added to this.

Concept 6 requires one capacitive seal sensor per moving object, giving two for the sunroof and one for each window. Volvo has received proposals of similar concepts from suppliers and they have all been disregarded as to expensive. Since no general price data could be found from the suppliers is a rough estimate done. The price for one capacitive seal for mass manufacturing is estimated to be the same as for a motor controller, giving concept 6 twice the cost compared to concept 2, 159.10 SEK.

For concept 12 is a minimum of one ultrasonic sensor needed for the panel, one for the curtain and one for each window. A similar concept was tested by Schlegl et. al. which used ultrasonic sensors from an existing automotive parking assist system (Schlegl et al., 2011). The cost estimation used here are based on a reference parking assistance system developed by TDK Product Center (TDK Corporation, 2020). Buying 1000 pieces of the components used by TDK would give a cost around 50 SEK, with an 80% price reduction due to mass manufacturing circumstances gives an estimated cost of 10 SEK per unit and a cost per vehicle of 139.55 SEK.

LiDAR technology is very advanced compared to the technologies used in the other concepts, the capacity of LiDAR sensors varies a lot and so does the price. Volvo are currently using LiDAR systems for a couple of functions, one of these functions is the breaking assistance. The retail price for buying one of these sensors as a spare part is 5673.22 SEK, “Teardown.com” have made an independent deep dive teardown of this breaking assistance sensor, the Volvo V40 31360888 LiDAR ID 223406-KCd. Based on the used components have they estimated a total cost for this LiDAR system of 22.61 dollars, equivalent to 218.76 SEK (based on the exchange rate at the time of writing, 1 USD = 9.68 SEK). A vehicle would require two sensors in order to monitor the entire inside of the vehicle, giving a total cost of 517 SEK per vehicle. (Cendrowicz, 2019)

Table 17 shows the estimated component cost per vehicle for the current solution and the remaining concepts. The table states what components that are needed in order to control the sunroof and all windows.

Table 17 Estimated component cost per vehicle

<table>
<tr><th colspan="4">Component costs base on prototyping (excluding base costs)</th></tr>
<tr><td>Concepts</td><td colspan="2">Components</td><td>Estimated component cost per vehicle</td></tr>
<tr><td rowspan="2">Current solutions</td><td>Sunroof</td><td>4 Relays or
2 Motor Controllers</td><td rowspan="2">113 /
79.55</td></tr>
<tr><td>Windows</td><td>4 Motor Controllers</td></tr>
<tr><td>Concept 1</td><td colspan="2">12 Relays or
6 Motor Controllers</td><td>180 /
79.55</td></tr>
<tr><td>Concept 2</td><td colspan="2">6 Motor Controllers</td><td>79.55</td></tr>
<tr><td>Concept 3</td><td colspan="2">12 Relays + 6 Ripple Counters Or
6 Motor Controllers + 6 Ripple Counters</td><td>237.2 /
136.75</td></tr>
<tr><td>Concept 4</td><td colspan="2">6 Motor Controllers + 6 Ripple Counters</td><td>136.75</td></tr>
<tr><td>Concept 6</td><td colspan="2">6 Motor Controllers +6 Capacitive sensors</td><td>159.10</td></tr>
<tr><td>Concept 12</td><td colspan="2">6 Motor Controllers + 6 ultrasonic sensors</td><td>139.55</td></tr>
<tr><td>Concept 16</td><td colspan="2">6 Motor Controllers + 2 LiDAR’s</td><td>517</td></tr>
</table>

9.4 Total Cost per Vehicle

To be able to compare the developed concepts with the current solution is the estimated development and calibration cost per vehicle combined with the estimated component cost per vehicle. For this comparison is it estimated that the development cost and the calibration cost is the same for all concepts. Table 18 shows the cost estimation for the current solution and all concepts.

Table 18 Estimated total cost per vehicle, including development, calibration and component costs

Estimated Total Cost Per Vehicle [SEK]			
Concepts	component cost	Development & Calibration cost	Total Cost
Current solutions	113 / 79.55	90.89	203.93/ 170.44
Concept 1	180 / 79.55	3.3	183.30/ 82.85
Concept 2	79.55	3.3	82.85
Concept 3	237.2 / 136.75	3.3	240.50/ 140.06
Concept 4	136.75	3.3	140.06
Concept 6	159.10	3.3	162.41
Concept 12	139.55	3.3	142.85
Concept 16	517	3.3	520.37

10 Concept Selection

In this chapter, the process of concept selection has been presented in two stages. Firstly, the screening of the remaining concepts against a reference solution and each other using the customer needs in a Pugh matrix. This is followed by the scoring stage where the concepts have been compared against each other using weighted criteria. After these stages, the results have been analyzed and the final recommendations have been made.

The knowledge gained during the prototyping and testing of the concepts combined with the cost estimations lay as the foundation for decision making in the concept selection phase. The concepts for which no working prototype could be built was the decisions based on the theory presented in the Technology Study. An overview of the remaining concepts and for which a working prototype could be built can be seen in Table 19.

Table 19 Show an overview of the 7 concepts taken into the Pugh matrix and for which a working prototype were build and what type of testing underwent

Concept Prototyping Overview		
Concept	Prototyping [✓/ ✗]	Prototype Testing [Formal/Informal/ No Testing]
1	✓	Formal & Informal
2	✓	Informal
3	✓	No Testing
4	✓	No Testing
6	✗	-
12	✗	-
16	✗	-

An elimination matrix, as shown in earlier chapters, and a Pugh matrix are used in the screening stage and a Kesselring matrix is used in the scoring stage. These matrices provide a structured method for evaluating concepts. They also visualize information that can support in decision making. Based on the results from the screening and scoring, one or a few concepts will be recommended for further development.

10.1 Concept Screening

The Pugh matrix is a method for unbiased concept screening. One concept is used as a reference solution and all the other concepts are then compared to this reference solution based on how well they fulfill the customer needs. The concepts are compared to the reference using a "+" if it performs better, a "-" if it performs worse and a "0" if it is like the reference. When all customer needs are evaluated, the scores are summed, and the concepts get a final score. The final score will be negative if the concept generally performs worse that the reference, zero if it is similar and positive if the overall performance is estimated to be better than the reference. This is done in iterations with different concepts as the reference solution. Normally a competing product or an earlier generation of the same product is set as reference in the first iteration. (Ulrich & Eppinger, 2012)

Three iterations of the Pugh matrix were performed. In the first iteration, the current solution for the sunroof was set as the reference. The current solution is an indirect system based on hall sensors without any type of speed control. It is very similar to concept 1 but the current solution does not meet the requirements of FMVSS No. 118. In the second iteration, concept 16 was used as reference, and concept 2 was used as the reference in the final iteration.

All concepts performed well in the first iteration. Only concept 12 got a score of 1, indicating that it would perform very similar to the reference. Concepts 1 and 6 got a score of 3 and the others scored even higher. Concept 16 scored best of all with 8, performing better than or equal to the reference for all criteria except cost where it got a "-". Since it performed so well, concept 16 was set as the reference in the second iteration.

In the second iteration, all concepts got a negative score and concept 6 performed the best with scoring -3. The others got a score of -6. It is worth noting that all concept performed better than the reference in the cost criterion.

For the final iteration, concept 2 was used as reference since the prototype of the concept performed well in the in the testing. This iteration gave mixed results. Concept 16 still scored the highest with 7, concepts 1 and 12 got negative results -2 and -3 respectively. Concept 3 scored 0 and concept 4 and 6 scored 2 each.

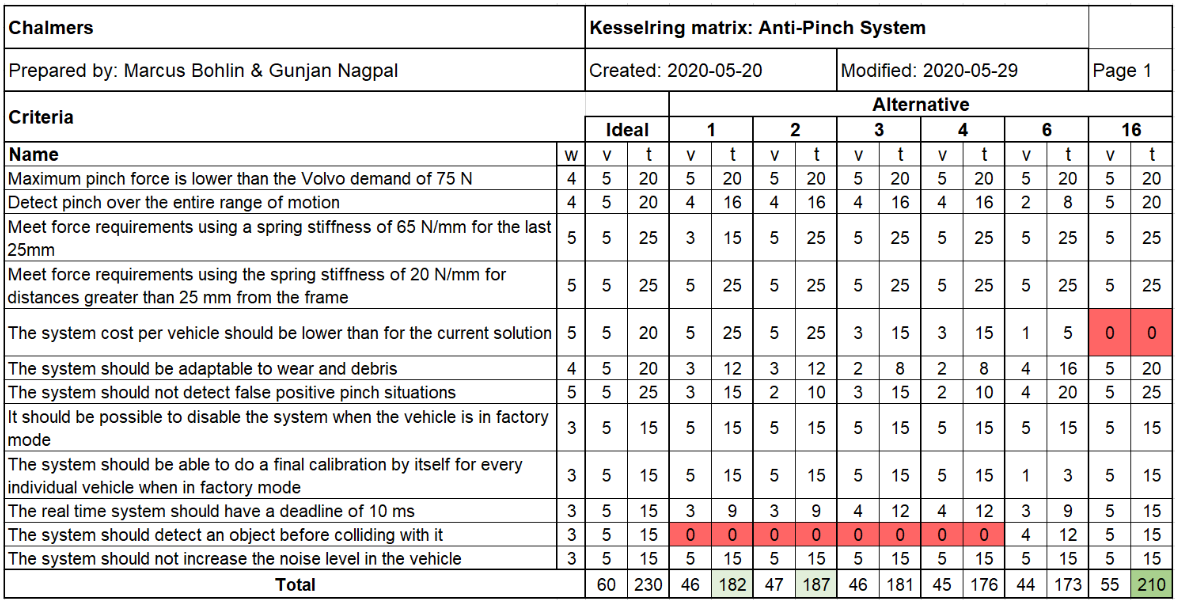
Based on the results from the three iterations, it was decided to eliminate concept 12. The concept scored lower than all the other concepts throughout the process. The remaining 6 concepts were taken further to the scoring process.

10.2 Concept Scoring

In the Pugh matrix, all the evaluation criteria have the same importance. The big difference with the Kesselring or scoring matrix is that the criteria are weighted based on the importance of the customer need they represent. A customer need with an importance of 5 in the customer needs list, is assigned a weighting of 5 in the Kesselring matrix. Similarly, a customer need with the importance of 3 get a weighting of 3. This entails that the more important needs have a greater influence on the final score of the concept. Another difference compared to the concept screening is that no reference concept is used. Every concept is scored completely based on its own performance. The performance of a concept against a specific criterion is ranked on a scale from 0-5, where 5 means that the concept completely fulfills or if possible, surpasses the criterion. A concept receives a score of 0 if it cannot in any way fulfill the criterion. Determining where on this scale that a concept is located is done through a discussion in the project group based on test results if there are any available, otherwise based on the theoretical knowledge about the concepts.

The scoring process has a couple of draw backs. Mainly that setting the importance of a customer need and determining how well a concept fulfils that need is extremely subjective if no concrete test results or extensive market analysis exists.

The concept that performed best in the scoring matrix was just like in the earlier stage concept 16 with an impressive margin of 23 points to concept 2 and 28 to concept 1 which ended up with scores of 187 respective 182 points. Concept 3 followed just behind with scores of 181. Concept 6 and 4 got the lowest scores with 173 and 176. The entire scoring matrix can be seen in Figure 60.



Chalmers		Kesselring matrix: Anti-Pinch System													
Prepared by: Marcus Bohlin & Gunjan Nagpal		Created: 2020-05-20						Modified: 2020-05-29						Page 1	
Criteria				Alternative											
		Ideal		1		2		3		4		6		16	
Name	w	v	t	v	t	v	t	v	t	v	t	v	t	v	t
Maximum pinch force is lower than the Volvo demand of 75 N	4	5	20	5	20	5	20	5	20	5	20	5	20	5	20
Detect pinch over the entire range of motion	4	5	20	4	16	4	16	4	16	4	16	2	8	5	20
Meet force requirements using a spring stiffness of 65 N/mm for the last 25mm	5	5	25	3	15	5	25	5	25	5	25	5	25	5	25
Meet force requirements using the spring stiffness of 20 N/mm for distances greater than 25 mm from the frame	5	5	25	5	25	5	25	5	25	5	25	5	25	5	25
The system cost per vehicle should be lower than for the current solution	5	5	20	5	25	5	25	3	15	3	15	1	5	0	0
The system should be adaptable to wear and debris	4	5	20	3	12	3	12	2	8	2	8	4	16	5	20
The system should not detect false positive pinch situations	5	5	25	3	15	2	10	3	15	2	10	4	20	5	25
It should be possible to disable the system when the vehicle is in factory mode	3	5	15	5	15	5	15	5	15	5	15	5	15	5	15
The system should be able to do a final calibration by itself for every individual vehicle when in factory mode	3	5	15	5	15	5	15	5	15	5	15	1	3	5	15
The real time system should have a deadline of 10 ms	3	5	15	3	9	3	9	4	12	4	12	3	9	5	15
The system should detect an object before colliding with it	3	5	15	0	0	0	0	0	0	0	0	4	12	5	15
The system should not increase the noise level in the vehicle	3	5	15	5	15	5	15	5	15	5	15	5	15	5	15
Total		60	230	46	182	47	187	46	181	45	176	44	173	55	210

Figure 60 Kesselring matrix

Based on the results from the prototype testing is it believed that both concept 1 and 2 will fulfill the customer needs related to the pinching forces. Since concept 3 and 4 are based on ripple counting technology, which should be even more sensitive are they also believed to meet these needs. Concept 6 and 16 can detect obstacles before colliding with them meaning that they automatically have a pinch force of 0 N. That three of the most important criteria are related to the pinch force which are met to a great extent by all concepts, inevitably leads to that the concepts receive similar scores. In fact, the scores are so similar, and the estimations of the importance of specific needs and how well they are fulfilled by the concepts are so uncertain. That no direct conclusions could be taken based solely on these results. Different concepts suit better for different scenarios, concept 16 for example outperforms the other concepts when its coms to technical performance. But it has a huge disadvantage due to its high cost, which is of outmost importance in the automotive industry. The recommendations for a suitable concept are elaborated further in the discussion chapter.

11 Discussion

In this chapter, general observations from the project, the concept recommendations and further development of these concepts has been discussed. The chapter ends with a short account on the patent situation and ethical aspects considered during the project.

11.1 Project

When the project started only a few guidelines were given by Volvo. Scope determination and project planning were carried out by the project team to a great extent. Since no similar system existed at Volvo, no previous knowledge existed that could be utilized. Instead, the work started from scratch by gathering information from research articles, patents, internet search etc. The basic technologies that an anti-pinch system is based on are relatively simple and well established, so the first steps in building simple prototypes could be taken after just a couple of weeks using the hardware that existed at Volvo. The fact that the prototyping could start at such an early stage into the project turned out to be extremely fortunate, since in response to the COVID-19 pandemic, Volvo Cars was forced to shut down temporarily and operate at reduced working hours. By the time the shutdown occurred, enough progress on the prototyping was already made, so the project proceeded without being affected to any great extent. However, some disturbance due to the shutdown occurred, mainly when it came to lead times for ordering parts or services within Volvo. One example of this was the ordering of the ripple counting boards, which were vital for building the prototypes of concepts 3 and 4. The ripple counting board is a reference design from Texas Instruments, but cannot be bought off the shelf. Instead the components must be bought separately and mounted on a PCB. The mounting of the components was done within Volvo. But it took just over nine weeks from the day of placing the order before the boards were received. In addition to this, none of the boards were fully functioning. This long lead time left a small amount of time for troubleshooting the problems with the boards and get the prototypes of concept 3 and 4 to work properly.

One thing that distinguished itself in this project compared to earlier projects that the group members have participated in at the university, was that the developed prototypes could be tested by an independent third party in accordance to a standardized testing procedure. Even if not all concepts were tested by this independent party, similar results were obtained and by comparing them to the results from the standardized testing, their credibility increased drastically.

11.2 Applicability for Multiple Applications

One of the main goals of the project was to show that one system could be implemented in multiple parts in a vehicle that required an anti-pinch system. Unfortunately, the limited time restricted the work to only be focused on the sunroof and the power windows, both of which utilize sliding motion. Most of the other applications like the tailgate are based on rotating motions, which could require different solutions. So, even if the time was the limiting factor for the project from investigating all areas, it was showed that a software developed for the sunroof panel could be used for the sunroof curtain and the power windows. Only small changes in the software, like changing the range of motion and creating a new threshold curve were enough to make it work in the different applications.

11.3 Concept Recommendation

The six concepts that made it through the entire development process were ranked very similarly to each other in both the screening and the scoring processes, except concept 16. This makes it hard to take a decision based on those results. The anti-pinch system must fulfill a few requirements regarding the pinching force stated by Volvo and the legal documents. If these requirements are fulfilled, there are none or diminishing returns to user value by having a more sensitive system. As stated earlier, the anti-pinch system only attracts the user's attention if it registers false pinches or if the system malfunctions and thereby causes an accident. Based on this, it would be a reasonable recommendation to use the simplest and cheapest solution that meets the requirements, referring to concept 1 or concept 2. And this is the recommendation that would be given by the project group to Volvo if they asked for a solution that could be implemented within one to two years. It must be noted that is not completely verified if the prototypes of concept 1 and 2 would meet the harsher FMVSS No. 118 requirement in a realistic environment since the concepts have only been tested in laboratory conditions. Some additional improvement through iterations with formal testing needs to be done in order to reduce the sensitivity of the systems to avoid false pinches.

If the concept were to be implemented in three to four years, the recommendation could look different. In that case, concept 3 or 4 would be more suitable. The cost of the components can decrease drastically and concept 3 and 4 would then provide a much more granular control compared to concept 1 and 2 at a similar cost.

A similar argument applies for concept 16 but for an even longer time horizon. Throughout the screening and scoring process concept 16 outperforms all the other concepts, but the cost per vehicle makes it difficult to justify the implementation of the concept. LiDAR technology is significantly newer and more advanced than the other technologies evaluated. Thanks to its important role within autonomous driving, a lot of research and development is being done within LiDAR technology. The technology is also starting to appear in consumer electronics like smartphones and tablets which will help in reducing the cost and increasing the performance. One of Volvo's partners within LiDAR technology is expects to be able to produce LiDAR sensors for around 3 USD per piece within a couple of years. This would change the cost equation drastically. Concept 16 also has the advantage that it could be used for more functions than just the anti-pinch and position control systems, spreading the cost between more functions. If the concept were to be implemented in more than seven years concept 16 would be recommended.

When it comes to concept 6 is the situation a bit different, the concept requires either hall sensors or ripple counting technology for the position control system, and then a specially made seal including capacitive sensors for the anti-pinch system. It is hard to argue for this concept when the first four concepts meet the requirements by only using hall sensors or ripple counting. The user value added by having the capacitive sensor for detecting a pinch situation is very limited, but the extra component and assembly cost is inevitable. Therefore, the concept 6 is not recommended for further development.

11.4 Further Development

One interesting area to look at in order to improve the concepts further, is to minimize the possibility for false pinches. One of these measures could be to disable the anti-pinch system in the sunroof panel if the curtain is already fully closed. This can be done since it is almost impossible to get pinched by the panel, from the inside of the vehicle if the curtain covers the entire opening in the sunroof. Another option could be to disable or drastically decrease the sensitivity of the anti-pinch system in the sunroof and front windows when the car is driving. The purpose of paragraph S5 in FMVSS, that is not fulfilled by the current solution, is to protect children that are inside a vehicle without a responsible vehicle operator being present. If the S5 requirements are fulfilled can the functions such as closing the windows or sunroof be operated from a distance, for example by using a mobile application. If the vehicle is driving, there is no longer any point in operating these functionalities from a distance since the driver obviously already is in the vehicle and would observe if a pinch situation is about to occur in the sunroof or the front windows. Implementing these solutions could drastically reduce the risk of false pinches without any major developmental effort.

Another area that could be improved without major developmental efforts is the calibration of the anti-pinch system. This is a very expensive and time-consuming activity that affected Volvo employees have expressed dissatisfaction with during the interviews. By creating simulation models of the sunroof and the windows, it should be possible to perform at least parts of the calibration using a software tool. This could have a great impact on the development time and the calibration costs, and reduce the frustration felt within the organization.

11.5 Patents

Two different patent analyses were done during the project, one in the early stage of the project and one in the later stages. The result was the same from both analyses, that a lot of patents existed for anti-pinch and position control systems. Most of them are very similar, stating that they rely on hall sensors or ripple counting technology to determine the position and speed of the window, sunroof or slide door, and then compare the speed to a threshold value to identify a pinch situation. It is difficult to distinguish the patents from each other and define what novelty they were protecting since they were so similar. Most of them were protecting a specific circuit set up which was irrelevant for this project since the aim has been on getting a proof of concept, and off the shelf circuits and components have been used throughout the project.

11.6 Ethical Considerations

Although the main function of an anti-pinch systems relates to the safety of passengers, there are a few ethical aspects that were considered as a part of the development process. The main underlying reason behind the introduction of the American and United Nations safety regulations has been injury incidents related to children. Furthermore, the most recent amendments and proposed amendments are also driven by factors that improve safety for children. By having more control over the operation of the anti-pinch system in different parts, Volvo Cars can make sure that they deliver the safest possible experience for children inside their vehicles. Contrary to this, if Volvo fails to ensure a safe-enough anti-pinch system for their vehicles, they risk customer disappointment, and in some extreme situations, fines and/or legal action. The Volvo Cars brand is known for its focus on safety. At present, if there happens to be a safety related incident with the anti-pinch system(s), it would be Volvo’s supplier that takes the responsibility since that is a part of the agreement with the supplier. This saves Volvo from legal consequences in the case of a severe accident related to the anti-pinch system, but the focus of the general public would most likely be on Volvo not the supplier. If Volvo were to implement their own anti-pinch system, they will have to bear all responsibility.

Some ethical questions could also be raised regarding concept 16 since it is always monitoring the inside of the vehicle. It is of privacy concern how the data from the LiDARs is stored and what it is used for. The data could be used for improving the system further, but it might also infringe on the passenger's privacy. This problem is relevant for all types of monitoring of the interior compartment but using LiDAR sensors might be a better alternative from a passenger privacy perspective compared to other types of image processing methods.

12 Conclusion

The purpose of the thesis was to show that one anti-pinch system could be developed and adapted to work with multiple applications, instead of buying separate systems from multiple suppliers.

The prototypes built during this project proves that within a short time and limited resources, an anti-pinch system that meets the ECE R21-01 requirements, and with high certainty also the FMVSS No. 118 requirements, can be developed. It was also demonstrated that one software developed for the sunroof panel could be implemented in the sunroof curtain and in the power windows with minimal modification. Based on this, one system can be used for multiple applications which would enable significant cost reductions relating to development and calibration. This showed the possibility of developing a system in-house bringing cost reductions since a single Volvo owned system can be used across all models instead of buying separate systems for each.

The developed concepts show that multiple different technologies can be used for an anti-pinch system and based on the test results it can be said that even the simplest of all the concepts has a realistic chance of meeting the strictest requirements. Five concepts suited for implementation at varying time scales were suggested. Two concepts based on hall sensors were suggested for an implementation within one to two years. Two concepts based on ripple counting technology were recommended for implementation within three to four years, and finally a concept based on LiDAR sensors was recommended for the time when the technology matures. In addition to these concepts, different methods of counteracting the current problems with false pinches were suggested. These methods could be combined with the developed concepts or implemented in the system currently in use.

The prototypes built during the project were only proof of concept in nature, as they required further refinement before they could become ready for implementation. The next steps in the development process would be to adjust the sensitivity of the system to minimize the risk for false pinches and implementing the software in a vehicle and performing tests in real world environments.

13 Bibliography

Allegro. (2013). *Unipolar Hall-Effect Sensor IC Basics*. https://www.allegromicro.com/en/Insights-and-Innovations/Technical-Documents/Hall-Effect-Sensor-IC-Publications/Unipolar-Hall-Effect-Sensor-IC-Basics

Allegro MicroSystems, L. (2013a). *Bipolar Switch Hall-Effect ICs*. Allegro MicroSystems, LLC. https://www.allegromicro.com/en/Insights-and-Innovations/Technical-Documents/Hall-Effect-Sensor-IC-Publications/Bipolar-Switch-Hall-Effect-ICs

Allegro MicroSystems, L. (2013b). *Unipolar Hall-Effect Sensor IC Basics*. Allegro MicroSystems, LLC. https://www.allegromicro.com/en/Insights-and-Innovations/Technical-Documents/Hall-Effect-Sensor-IC-Publications/Unipolar-Hall-Effect-Sensor-IC-Basics

Allodi, M., Broggi, A., Giaquinto, D., Patander, M., & Prioletti, A. (2016). Machine learning in tracking associations with stereo vision and lidar observations for an autonomous vehicle. *2016 IEEE Intelligent Vehicles Symposium (IV)*, 648–653. https://doi.org/10.1109/IVS.2016.7535456

Almefelt, L. (2019). *Systematic Design - Overview* (p. 69).

Amin, I., & Beard, P. (2016). *Relay Replacement for Brushed DC Motor Drive in Automotive Applications*. Texas Instruments. https://www.ti.com/lit/pdf/slva837

ATMEL Corporation. (2016). *AVR480: Anti-Pinch System for Electrical Window*. http://ww1.microchip.com/downloads/en/AppNotes/doc7559.pdf

Biezen, M. van (Loyola M. U. (2015). *Special Topic - The Kalman Filter*. ILectureOnline. http://www.ilectureonline.com/lectures/subject/SPECIAL TOPICS/26/190

Cendrowicz, K. (2019, July). Deep Dive Teardown of Volvo LiDAR 31360888 Brake Assist Sensor. *TechInsights*. https://www.techinsights.com/products/ddt-1709-805

Consoli, A., Bottiglieri, A., Letor, R., Ruggeri, R., Testa, A., & Caro, S. de. (2004). Sensorless position control of DC actuators for automotive applications. *Conference Record of the 2004 IEEE Industry Applications Conference, 2004. 39th IAS Annual Meeting.*, *2*, 1217–1224 vol.2. https://doi.org/10.1109/IAS.2004.1348568

Cornell Law School. (n.d.). *Legal Information Institute*. https://www.law.cornell.edu/cfr/text/49/571.118

Ghosh, M., Saha, P. K., & Panda, G. K. (2018). Hybrid Computational Mechanical Sensorless Fuzzified Technique for Speed Estimation of Permanent Magnet Direct Current Brushed Motor. *IEEE Transactions on Industrial Electronics*, *65*(6), 4565–4573. https://doi.org/10.1109/TIE.2017.2767553

Hanselman, D. (2006). *Brushless Permanent Magnet Motor Design*. Magna Physics Publishing.

Hirzel, T. (n.d.). *Arduino - PWM*. Arduino.Cc. Retrieved May 26, 2020, from https://www.arduino.cc/en/Tutorial/PWM

History of the CAN technology. (n.d.). CAN-CIA. Retrieved May 20, 2020, from https://www.can-cia.org/can-knowledge/can/can-history/

Jost, D. (n.d.). *What is an IR sensor*. FIRCE Electronics. https://www.fierceelectronics.com/sensors/what-ir-sensor

LIN Steering Group. (n.d.). LIN Steering Group. Retrieved May 20, 2020, from www.lin-subbus.org

Mellah, H., Hemsas, K. E., Taleb, R., & Cecati, C. (2018). Estimation of speed, armature temperature, and resistance in brushed DC machines using a CFNN based on BFGS BP. *Turkish Journal of Electrical Engineering & Computer Sciences*, *26*(6), 3181–3191. https://doi.org/10.3906/elk-1711-330

Microchip Technology Inc. (2004). *Brushed DC Motor Fundamentals*. http://fritzing.org/media/fritzing-repo/projects/c/cmn-fan-control/other_files/Brushed DC Motor Fundamentals.pdf

National Highway Traffic Safety Administration. (2009). *Federal Motor Vehicle Safety Standards: Power-operated window, partition, and roof panel systems*. https://www.federalregister.gov/documents/2009/09/01/E9-21042/federal-motor-vehicle-safety-standards-power-operated-window-partition-and-roof-panel-systems

National Highway Traffic Safety Administration. (2011). *Standard No 118; Power-operated window, partion, and roof panel systems*. https://www.govinfo.gov/app/details/CFR-2011-title49-vol6/CFR-2011-title49-vol6-sec571-118

Nelson, R. R. (2007). IT Project Management: Infamous Failures, Classic Mistakes, and Best Practices. *MIS Quarterly Executive, 6*, 74.

Nitsche, B., & Herrmann, R. (2009). Direct Sensor Solutions for Anti Pinch and Collision Avoidance for Motorized Closures. *SAE Technical Paper 2009-01-0637*. https://doi.org/10.4271/2009-01-0637

Pak, J. M., Kang, S. J., Pae, D. S., & Lim, M. T. (2017). Accurate pinch detection using recent finite measurements for automotive anti-pinch sunroof systems. *International Journal of Control, Automation and Systems*, *15*(5), 2443–2447. https://doi.org/10.1007/s12555-016-0328-8

Rajaram, S., & Murugesan, S. (1978). A New Method for Speed Measurement/Control of DC Motors. *IEEE Transactions on Instrumentation and Measurement*, *27*(1), 99–102. https://doi.org/10.1109/TIM.1978.4314629

Rylander, A., & Wallin, E. (2003). *LIN - Local Interconnect Network - for use as sub-bus in Volvo Trucks*.

Schlegl, T., Bretterklieber, T., Neumayer, M., & Zangl, H. (2011). Combined Capacitive and Ultrasonic Distance Measurement for Automotive Applications. *IEEE Sensors Journal*, *11*(11), 2636–2642. https://doi.org/10.1109/JSEN.2011.2155056

sen M. Kuo, Bob H. Lee, and W. T. (2013). *Real-Time Digital Signal Processing : Fundamentals, Implementations and Applications* (3rd ed.). John Wiley & Sons, Incorporated.

Simons-Morton, B., Lerner, N., & Singer, J. (2005). The observed effects of teenage passengers on the risky driving behavior of teenage drivers. *Accident Analysis & Prevention*, *37*(6), 973–982. https://doi.org/10.1016/j.aap.2005.04.014

Sollmann, M., Schurr, G., Duffy-Baumgaertner, D., & Huck, C. (2004). Anti Pinch Protection for Power Operated Features. *SAE 2004 World Congress & Exhibition*. https://doi.org/10.4271/2004-01-1108

Storr, W. (n.d.). *Electronics Tutorials - Hall Effect Sensors*. AspenCore Electronics Tutorials. Retrieved January 22, 2020, from https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html

Storr, W. (2014). *Passive Low Pass Filter*. AspenCore Electronics Tutorials. https://www.electronics-tutorials.ws/filter/filter_2.html

Sullivan, M. (2017). *The seat remembers: Brushed DC motor ripple counting drives innovation in full-featured memory seats* (No. 199; Behind the Wheel). https://e2e.ti.com/blogs_/b/behind_the_wheel/archive/2017/08/16/the-seat-remembers-brushed-dc-motor-ripple-counting-drives-innovation-in-full-featured-memory-seats

TDK Corporation. (2020). *Ultrasonic Parking Sensors for Automated Parking*. https://product.tdk.com/info/en/products/sensor/ultrasonic/sensor-disk/technote/apn_parking-assist.html

Terzic, E., Terzic, J., Nagarajah, R., & Alamgir, M. (2012). Capacitive Sensing Technology. In *A Neural Network Approach to Fluid Quantity Measurement in Dynamic Environments* (pp. 11–37). Springer. https://doi.org/10.1007/978-1-4471-4060-3_2

Testa, A., de Caro, S., Scimone, T., & Letor, R. (2014). Pulse Counting Sensorless Detection of the Shaft Speed and Position of DC Motor Based Electromechanical Actuators. *Journal of Power Electronics*, *14*(5), 957–966. https://doi.org/10.6113/JPE.2014.14.5.957

Texas Instruments. (n.d.). *DRV8703-Q1 Automotive Brushed DC Gate Driver Evaluation Module*. Retrieved March 3, 2020, from https://www.ti.com/tool/DRV8703-Q1EVM

Texas Instruments. (2018). *Automotive Brushed-Motor Ripple Counter Reference Design for Sensorless Position Measurement*. https://www.ti.com/tool/TIDA-01421#0

Ulrich, K. T., & Eppinger, S. D. (2012). *Product Design and Development* (5th ed.).

Volvo Cars Group. (2020). *Volvo Cars Group*. https://group.volvocars.com/company

Watters, A. (2015). *Lego Mindstorms: A History of Educational Robots*.

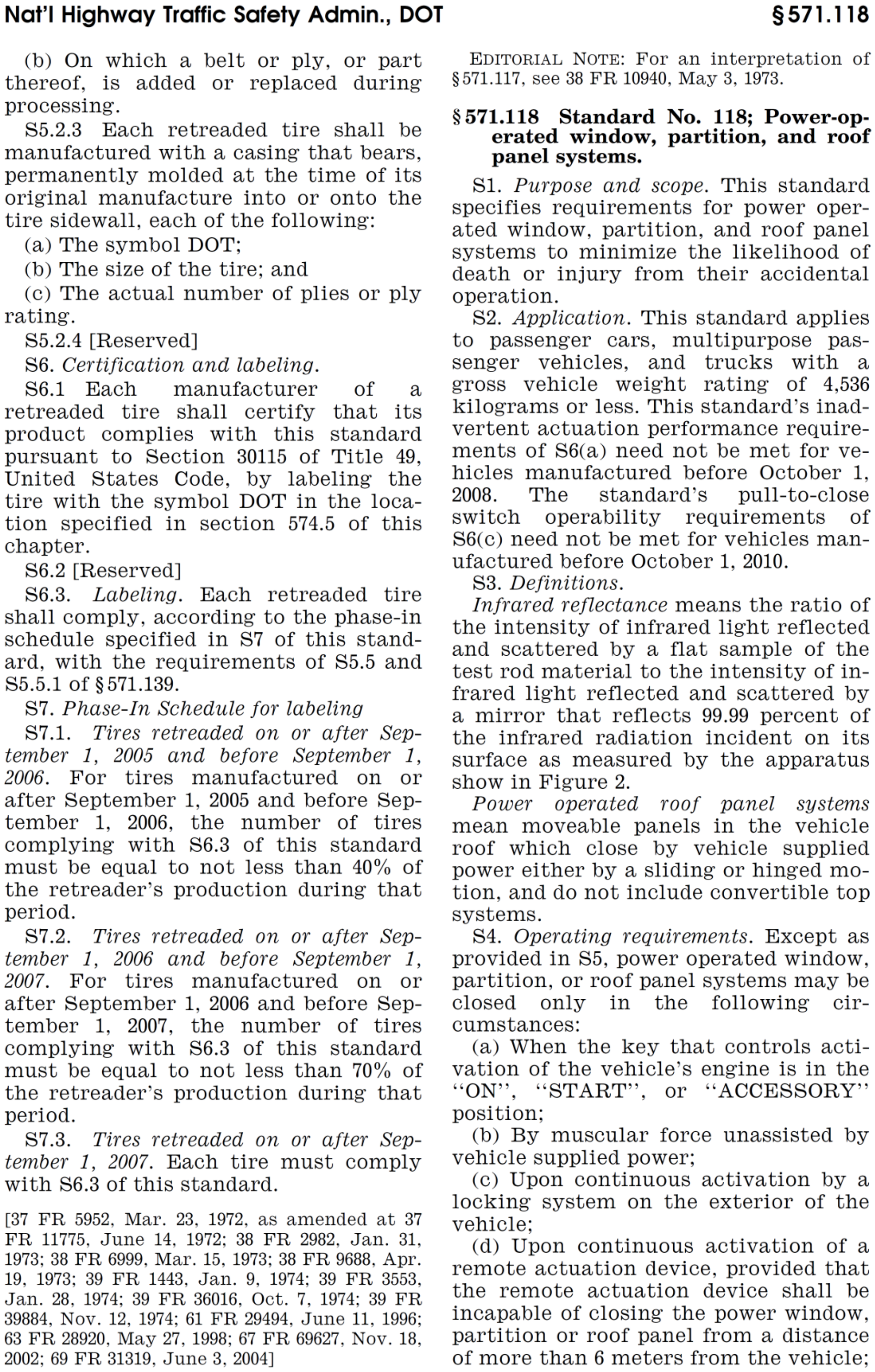
Webster, J. G., & Eren, H. (2017). *Measurement, Instrumentation, and Sensors Handbook: Electromagnetic, Optical, Radiation, Chemical, and Biomedical Measurement* (2nd Editio). CRC Press. https://www.routledge.com/p/book/9781138072176

Westgate, C. R., & Chien, C. L. (1980). *The Hall effect and its applications*. Springer, Boston, MA.

Zarchan, P., & Musoff, H. (2000). *Fundamentals of kalman filtering : A practical approach* (F. K. Lu, Ed.). American Institute of Aeronautics and Astronautics.

Appendix A

FMVSS Legal Document



(b) On which a belt or ply, or part thereof, is added or replaced during processing.

S5.2.3 Each retreaded tire shall be manufactured with a casing that bears, permanently molded at the time of its original manufacture into or onto the tire sidewall, each of the following:

(a) The symbol DOT;

(b) The size of the tire; and

(c) The actual number of plies or ply rating.

S5.2.4 [Reserved]

S6. *Certification and labeling.*

S6.1 Each manufacturer of a retreaded tire shall certify that its product complies with this standard pursuant to Section 30115 of Title 49, United States Code, by labeling the tire with the symbol DOT in the location specified in section 574.5 of this chapter.

S6.2 [Reserved]

S6.3. *Labeling.* Each retreaded tire shall comply, according to the phase-in schedule specified in S7 of this standard, with the requirements of S5.5 and S5.5.1 of §571.139.

S7. *Phase-In Schedule for labeling*

S7.1. *Tires retreaded on or after September 1, 2005 and before September 1, 2006.* For tires manufactured on or after September 1, 2005 and before September 1, 2006, the number of tires complying with S6.3 of this standard must be equal to not less than 40% of the retreader's production during that period.

S7.2. *Tires retreaded on or after September 1, 2006 and before September 1, 2007.* For tires manufactured on or after September 1, 2006 and before September 1, 2007, the number of tires complying with S6.3 of this standard must be equal to not less than 70% of the retreader's production during that period.

S7.3. *Tires retreaded on or after September 1, 2007.* Each tire must comply with S6.3 of this standard.

[37 FR 5952, Mar. 23, 1972, as amended at 37 FR 11775, June 14, 1972; 38 FR 2982, Jan. 31, 1973; 38 FR 6999, Mar. 15, 1973; 38 FR 9688, Apr. 19, 1973; 39 FR 1443, Jan. 9, 1974; 39 FR 3553, Jan. 28, 1974; 39 FR 36016, Oct. 7, 1974; 39 FR 39884, Nov. 12, 1974; 61 FR 29494, June 11, 1996; 63 FR 28920, May 27, 1998; 67 FR 69627, Nov. 18, 2002; 69 FR 31319, June 3, 2004]

EDITORIAL NOTE: For an interpretation of §571.117, see 38 FR 10940, May 3, 1973.

§571.118 Standard No. 118; Power-operated window, partition, and roof panel systems.

S1. *Purpose and scope.* This standard specifies requirements for power operated window, partition, and roof panel systems to minimize the likelihood of death or injury from their accidental operation.

S2. *Application.* This standard applies to passenger cars, multipurpose passenger vehicles, and trucks with a gross vehicle weight rating of 4,536 kilograms or less. This standard's inadvertent actuation performance requirements of S6(a) need not be met for vehicles manufactured before October 1, 2008. The standard's pull-to-close switch operability requirements of S6(c) need not be met for vehicles manufactured before October 1, 2010.

S3. *Definitions.*

Infrared reflectance means the ratio of the intensity of infrared light reflected and scattered by a flat sample of the test rod material to the intensity of infrared light reflected and scattered by a mirror that reflects 99.99 percent of the infrared radiation incident on its surface as measured by the apparatus show in Figure 2.

Power operated roof panel systems mean moveable panels in the vehicle roof which close by vehicle supplied power either by a sliding or hinged motion, and do not include convertible top systems.

S4. *Operating requirements.* Except as provided in S5, power operated window, partition, or roof panel systems may be closed only in the following circumstances:

(a) When the key that controls activation of the vehicle's engine is in the ''ON'', ''START'', or ''ACCESSORY'' position;

(b) By muscular force unassisted by vehicle supplied power;

(c) Upon continuous activation by a locking system on the exterior of the vehicle;

(d) Upon continuous activation of a remote actuation device, provided that the remote actuation device shall be incapable of closing the power window, partition or roof panel from a distance of more than 6 meters from the vehicle;

(e) During the interval between the time the locking device which controls the activation of the vehicle's engine is turned off and the opening of either of a two-door vehicle's doors or, in the case of a vehicle with more than two doors, the opening of either of its front doors;

(f) If the window, partition, or roof panel is in a static position before starting to close and in that position creates an opening so small that a 4 mm diameter semi-rigid cylindrical rod cannot be placed through the opening at any location around its edge in the manner described in S5(b); or

(g) Upon continuous activation of a remote actuation device, provided that the remote actuation device shall be incapable of closing the power window, partition or roof panel if the device and the vehicle are separated by an opaque surface and provided that the remote actuation device shall be incapable of closing the power window, partition or roof panel from a distance of more than 11 meters from the vehicle.

S5. *Automatic reversal systems.* A power-operated window, partition, or roof panel system that is capable of closing or of being closed under any circumstances other than those specified in S4 shall meet the requirements of S5.1, S5.2, and, if applicable, S5.3.

S5.1. While closing, the power-operated window, partition, or roof panel shall stop and reverse direction either before contacting a test rod with properties described in S8.2 or S8.3, or before exerting a squeezing force of 100 newtons (N) or more on a semi-rigid cylindrical test rod with the properties described in S8.1, when such test rod is placed through the window, partition, or roof panel opening at any location in the manner described in the applicable test under S7.

S5.2. Upon reversal, the power-operated window, partition, or roof panel system must open to one of the following positions, at the manufacturer's option:

(a) A position that is at least as open as the position at the time closing was initiated;

(b) A position that is not less than 125 millimeters (mm) more open than the position at the time the window reversed direction; or

(c) A position that permits a semi-rigid cylindrical rod that is 200 mm in diameter to be placed through the opening at the same location as the rod described in S7.1 or S7.2(b).

S5.3. If a vehicle uses proximity detection by infrared reflection to stop and reverse a power-operated window, partition, or roof panel, the infrared source shall project infrared light at a wavelength of not less than 850 nm and not more than 1050 nm. The system shall meet the requirements in S5.1 and S5.2 in all ambient light conditions from total darkness to 64,500 lux (6,000 foot candles) incandescent light intensity.

S6 *Actuation Devices.* Except as provided in paragraph S6(b), actuation devices in the occupant compartments of vehicles used to close power-operated windows, partitions, and roof panels must meet the following requirements:

(a) An actuation device must not cause a window, partition, or roof panel to begin to close from any open position when tested as follows:

(1) Using a stainless steel sphere having a surface finish between 8 and 4 micro inches and a radius of 20 mm ±0.2 mm, place the surface of the sphere against any portion of the actuation device.

(2) Apply a force not to exceed 135 Newtons (30 pounds) through the geometric center of the sphere. This force may be applied at any angle with respect to the actuation device.

(3) For actuation devices that cannot be contacted by the sphere specified in S6(a)(1) prior to the application of force, apply a force up to the level specified in S6(a)(2) at any angle in an attempt to make contact with the actuation device. The sphere is directionally applied in such a manner that, if unimpeded, it would make contact with the actuation device.

(b) The requirement in S6(a) does not apply to either—

(1) actuation devices that are mounted in a vehicle's roof, headliner, or overhead console that can close power-operated windows, partitions, or roof panels only by continuous rather than momentary switch actuation, or

(2) actuation devices for closing power-operated windows, partitions, or

roof panels which comply with paragraph S5.

(c) Any actuation device for closing a power-operated window must operate by pulling away from the surface in the vehicle on which the device is mounted. An actuation device for closing a power-operated window must operate only when pulled vertically up (if mounted on the top of a horizontal surface), or out (if mounted on a vertical surface), or down (if mounted on the underside of an overhead surface), or in a direction perpendicular to the surrounding surface if mounted in a sloped orientation, in order to cause the window to move in the closing direction.

S7. *Test procedures.*

S7.1. *Test procedure for testing power-operated window, partition, or roof panel systems designed to detect obstructions by physical contact or by light beam interruption:* Place the test rod of the type specified in S8.1 or S8.2, as appropriate, through the window, partition, or roof panel opening from the inside of the vehicle such that the cylindrical surface of the rod contacts any part of the structure with which the window, partition, or roof panel mates. Typical placements of test rods are illustrated in Figure 1. Attempt to close the power window, partition, or roof panel by operating the actuation device provided in the vehicle for that purpose.

S7.2. *Test procedure for testing power-operated window, partition, or roof panel systems designed to detect the proximity of obstructions using infrared reflectance:*

(a) Place the vehicle under incandescent lighting that projects 64,500 lux (6,000 foot candles) onto the infrared sensor. The light is projected onto the infrared sensor by aiming the optical axis of a light source outside the vehicle as perpendicular as possible to the lens of the infrared sensor. The intensity of light is measured perpendicular to the plane of the lens of the infrared sensor, as close as possible to the center of the lens of the infrared sensor.

(b) Place a test rod of the type specified in S8.3 in the window, partition, or roof panel opening, with the window, partition, or roof panel in any position. While keeping the rod stationary, attempt to close the window, partition, or roof panel by operating the actuation device provided in the vehicle for that purpose. Remove the test rod. Fully open the window, partition, or roof panel, and then begin to close it. While the window, partition, or roof panel is closing, move a test rod so that it approaches and ultimately extends through (if necessary) the window, partition, or roof panel opening, or its frame, in any orientation from the interior of the vehicle. For power partitions that have occupant compartment space on both sides of the partition, move the test rod into the partition opening from either side of the partition.

(c) Repeat the steps in S7.2(a) and (b) with other ambient light conditions within the range specified in S5.3.

S8. *Test rods.*

S8.1. *Rods for testing systems designed to detect obstructions by physical contact:*

(a) Each test rod is of cylindrical shape with any diameter in the range from 4 mm to 200 mm and is of sufficient length that it can be hand-held during the test specified in S7 with only the test rod making any contact with any part of the window, partition, or roof panel or mating surfaces of the window, partition, or roof panel.

(b) Each test rod has a force-deflection ratio of not less than 65 N/mm for rods 25 mm or smaller in diameter, and not less than 20 N/mm for rods larger than 25 mm in diameter.

S8.2. *Rods for testing systems designed to detect obstructions by light beam interruption:* Each test rod has the shape and dimensions specified in S8.1 and is, in addition, opaque to infrared, visible, and ultraviolet light.

S8.3. *Rods for testing systems designed to detect the proximity of obstructions using infrared reflection:*

(a) Each rod is constructed so that its surface has an infrared reflectance of not more than 1.0 percent when measured by the apparatus in Figure 2, in accordance with the procedure in S9.

(b) Each rod has the shape and dimensions specified in Figure 3.

S9. *Procedure for measuring infrared reflectance of test rod surface material.*

(a) The infrared reflectance of the rod surface material is measured using a flat sample and an infrared light source and sensor operating at a wavelength of 950 ±100 nm.

(b) The intensity of incident infrared light is determined using a reference mirror of nominally 100 percent reflectance mounted in place of the sample in the test apparatus in Figure 2.

(c) Infrared reflectance measurements of each sample of test rod surface material and of the reference mirror are corrected to remove the contribution of infrared light reflected an scattered by the sample holder an other parts of the apparatus befor computation of the infrared reflectanc ratio.

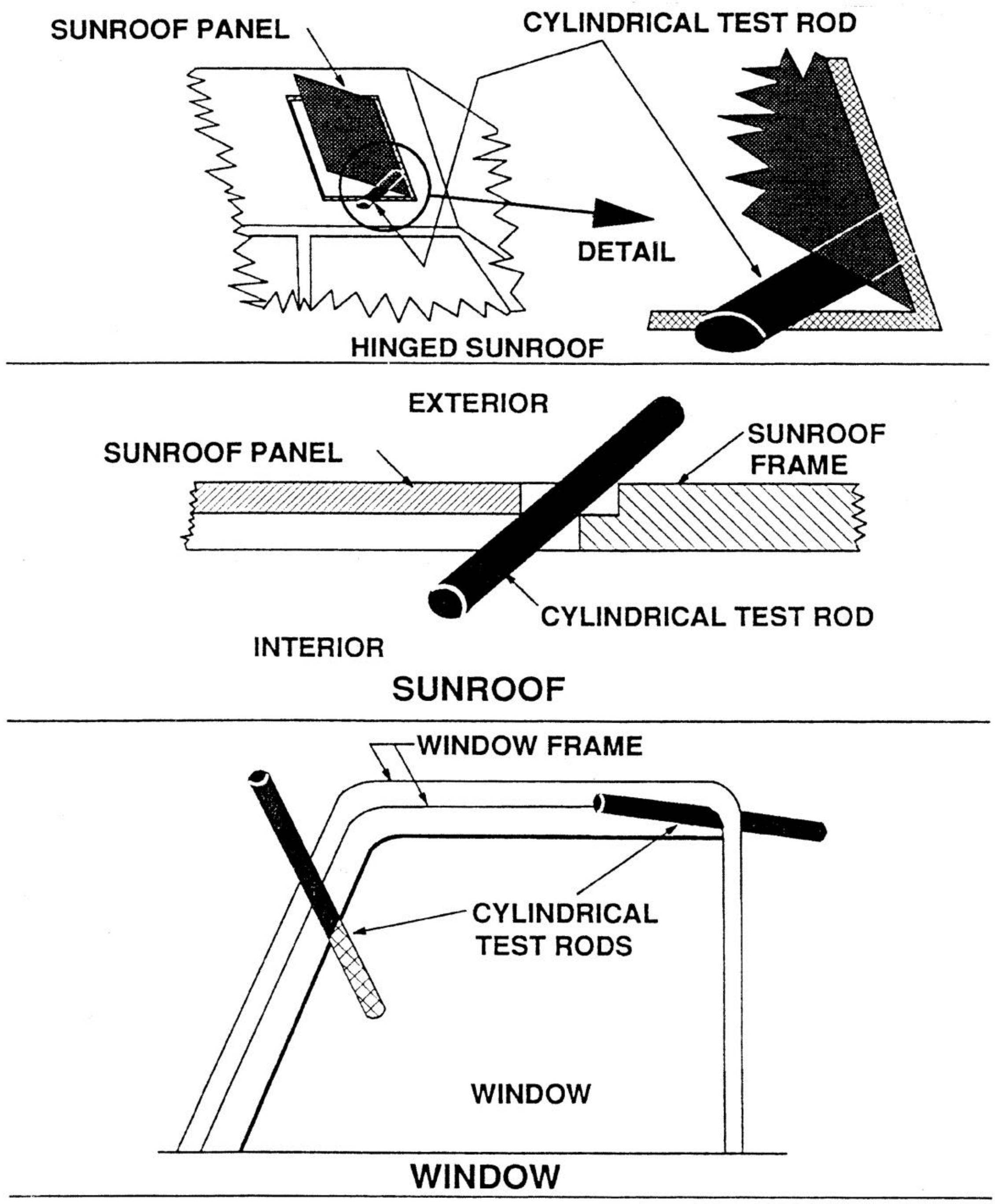
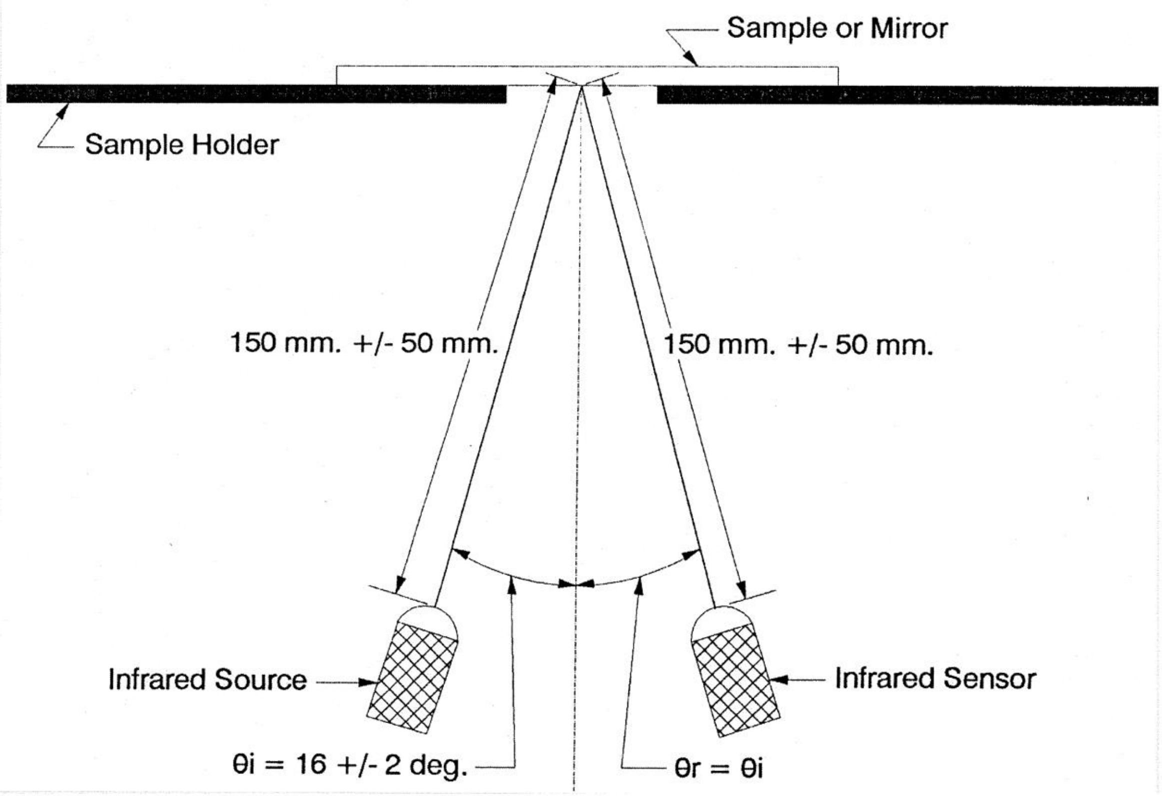


Figure 1 - Typical Cylindrical Test Rods Protruding through Sunroof and Window Daylight Openings

FIGURE 2 - REFLECTANCE TEST APPARATUS



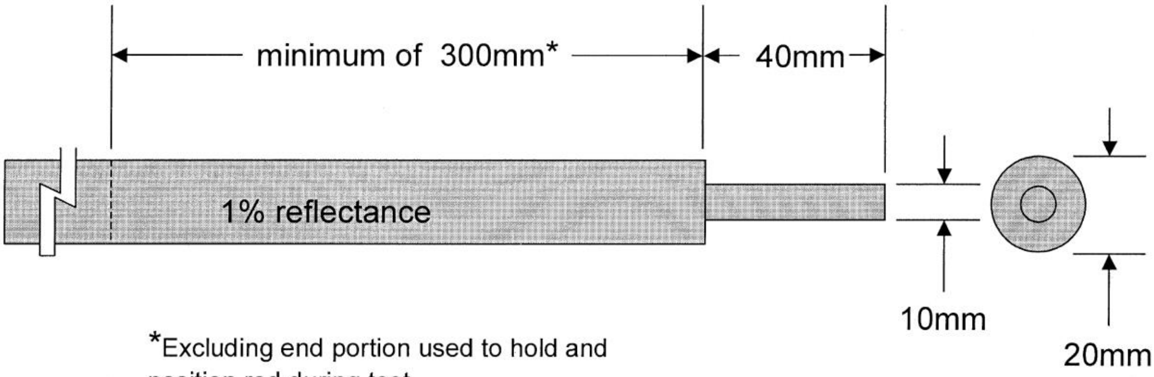

minimum of 300mm*
40mm
1% reflectance
10mm
20mm
*Excluding end portion used to hold and position rod during test

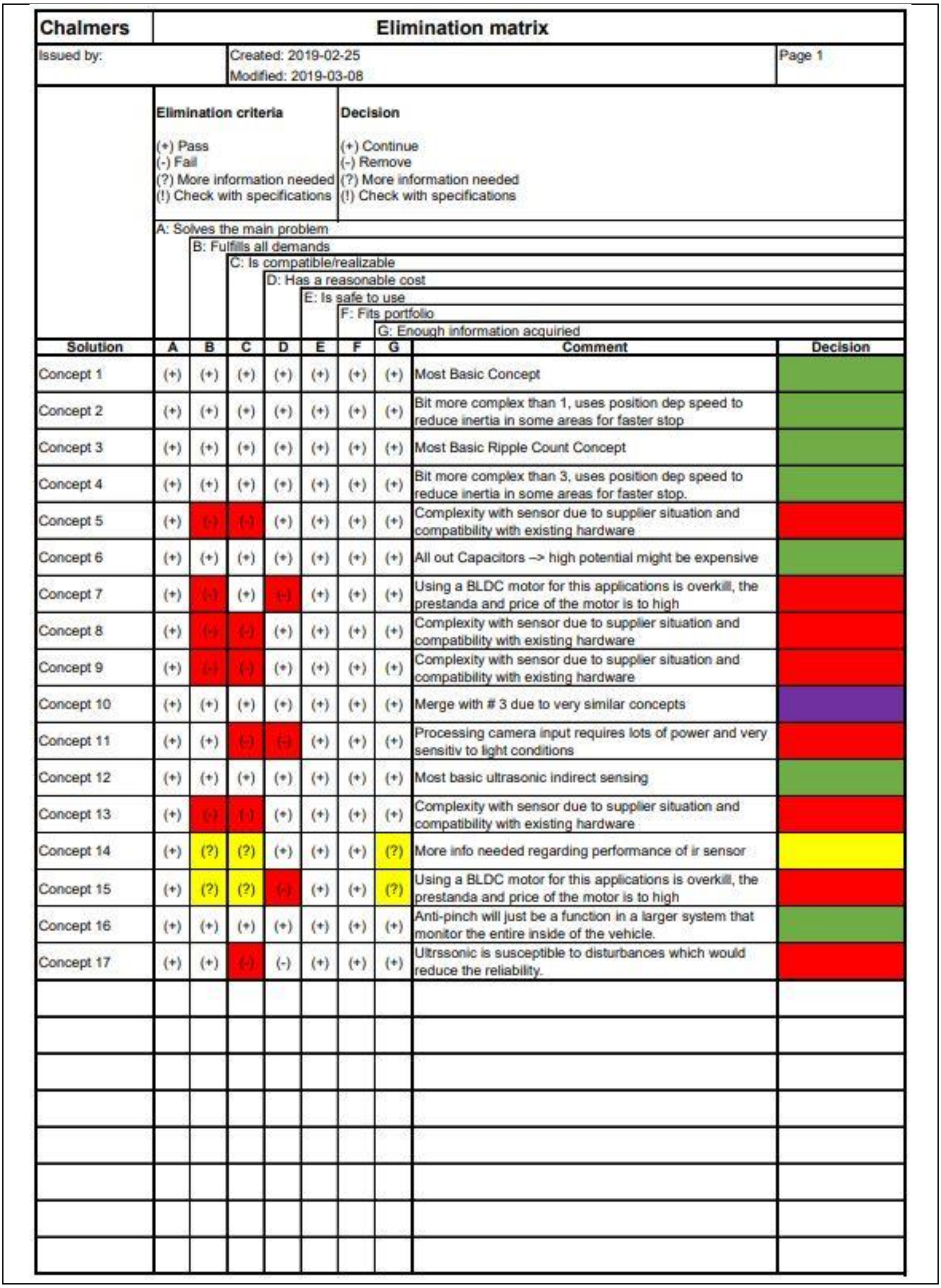
Figure 3

Cylindrical Rod

for Testing Non-Contact Infrared Reflection Systems

Appendix B

The elimination matrix



Chalmers	**Elimination matrix**	
Issued by:	Created: 2019-02-25 Modified: 2019-03-08	Page 1

Elimination criteria

(+) Pass
(-) Fail
(?) More information needed
(!) Check with specifications

Decision

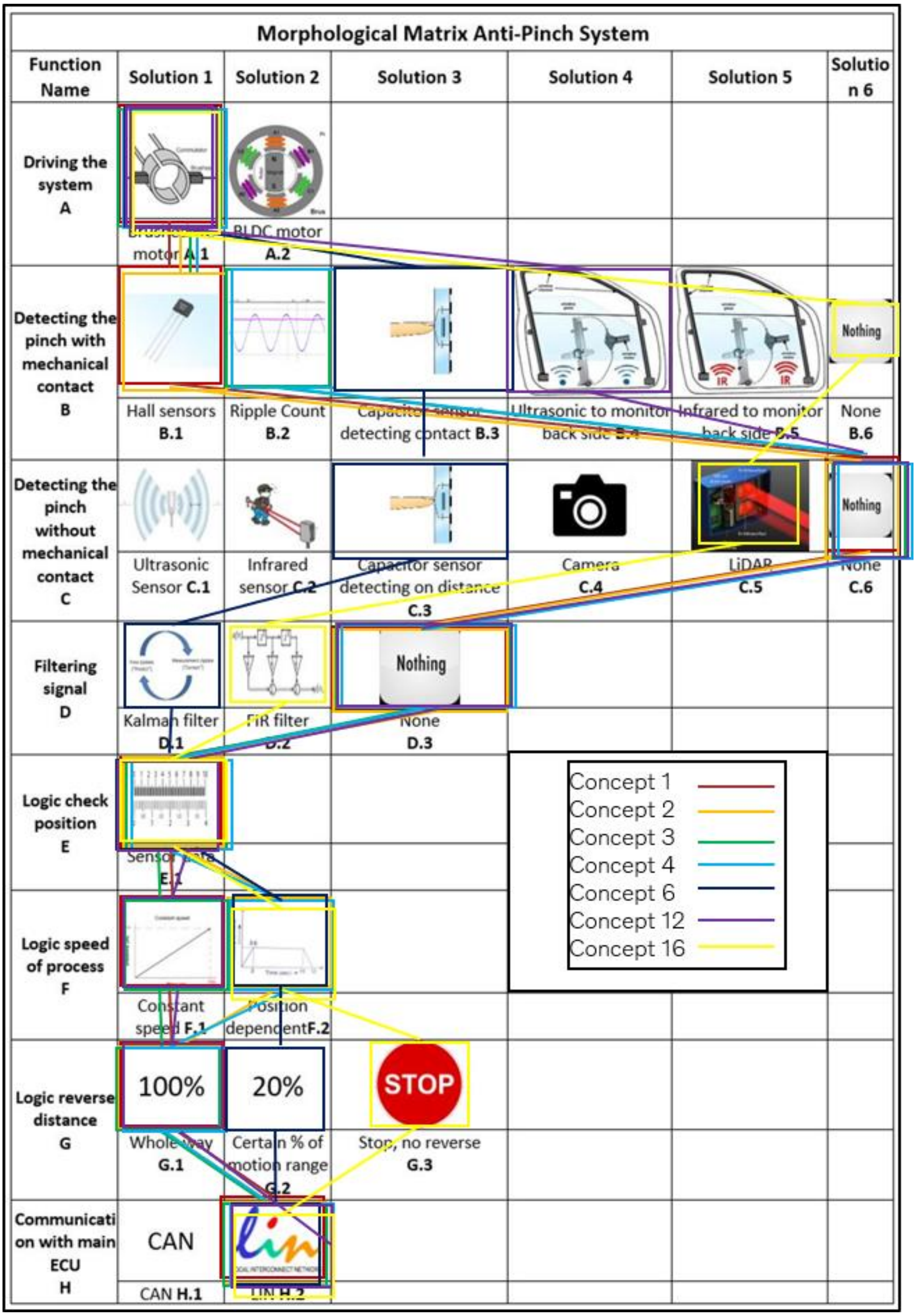
(+) Continue
(-) Remove
(?) More information needed
(!) Check with specifications

A: Solves the main problem
B: Fulfills all demands
C: Is compatible/realizable
D: Has a reasonable cost
E: Is safe to use
F: Fits portfolio
G: Enough information acquiried

Solution	A	B	C	D	E	F	G	Comment	Decision
Concept 1	(+)	(+)	(+)	(+)	(+)	(+)	(+)	Most Basic Concept	🟩
Concept 2	(+)	(+)	(+)	(+)	(+)	(+)	(+)	Bit more complex than 1, uses position dep speed to reduce inertia in some areas for faster stop	🟩
Concept 3	(+)	(+)	(+)	(+)	(+)	(+)	(+)	Most Basic Ripple Count Concept	🟩
Concept 4	(+)	(+)	(+)	(+)	(+)	(+)	(+)	Bit more complex than 3, uses position dep speed to reduce inertia in some areas for faster stop.	🟩
Concept 5	(+)	(-)	(-)	(+)	(+)	(+)	(+)	Complexity with sensor due to supplier situation and compatibility with existing hardware	🟥
Concept 6	(+)	(+)	(+)	(+)	(+)	(+)	(+)	All out Capacitors –> high potential might be expensive	🟩
Concept 7	(+)	(-)	(+)	(-)	(+)	(+)	(+)	Using a BLDC motor for this applications is overkill, the prestanda and price of the motor is to high	🟥
Concept 8	(+)	(-)	(-)	(+)	(+)	(+)	(+)	Complexity with sensor due to supplier situation and compatibility with existing hardware	🟥
Concept 9	(+)	(-)	(-)	(+)	(+)	(+)	(+)	Complexity with sensor due to supplier situation and compatibility with existing hardware	🟥
Concept 10	(+)	(+)	(+)	(+)	(+)	(+)	(+)	Merge with # 3 due to very similar concepts	🟪
Concept 11	(+)	(+)	(-)	(-)	(+)	(+)	(+)	Processing camera input requires lots of power and very sensitiv to light conditions	🟥
Concept 12	(+)	(+)	(+)	(+)	(+)	(+)	(+)	Most basic ultrasonic indirect sensing	🟩
Concept 13	(+)	(-)	(-)	(+)	(+)	(+)	(+)	Complexity with sensor due to supplier situation and compatibility with existing hardware	🟥
Concept 14	(+)	(?)	(?)	(+)	(+)	(+)	(?)	More info needed regarding performance of ir sensor	🟨
Concept 15	(+)	(?)	(?)	(-)	(+)	(+)	(?)	Using a BLDC motor for this applications is overkill, the prestanda and price of the motor is to high	🟥
Concept 16	(+)	(+)	(+)	(+)	(+)	(+)	(+)	Anti-pinch will just be a function in a larger system that monitor the entire inside of the vehicle.	🟩
Concept 17	(+)	(+)	(-)	(-)	(+)	(+)	(+)	Ultrssonic is susceptible to disturbances which would reduce the reliability.	🟥

Appendix C

Morphological matrix Figure 61 with the paths of the concepts remaining after the elimination matrix marked out.

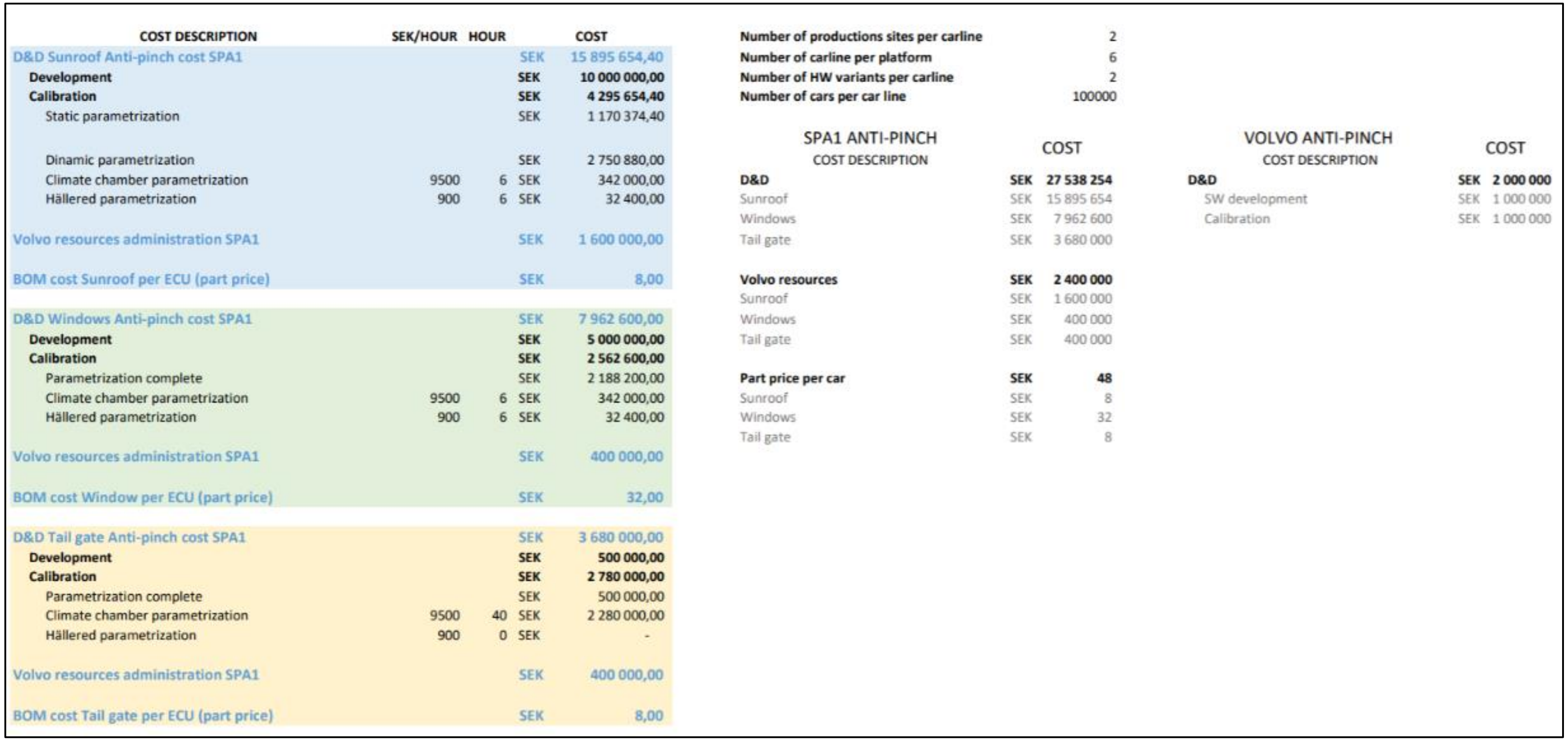


Morphological Matrix Anti-Pinch System						
Function Name	**Solution 1**	**Solution 2**	**Solution 3**	**Solution 4**	**Solution 5**	**Solution 6**
Driving the system A	Brushed DC motor **A.1**	BLDC motor **A.2**				
Detecting the pinch with mechanical contact B	Hall sensors **B.1**	Ripple Count **B.2**	Capacitor sensor detecting contact **B.3**	Ultrasonic to monitor back side **B.4**	Infrared to monitor back side **B.5**	Nothing None **B.6**
Detecting the pinch without mechanical contact C	Ultrasonic Sensor **C.1**	Infrared sensor **C.2**	Capacitor sensor detecting on distance **C.3**	Camera **C.4**	LiDAR **C.5**	Nothing None **C.6**
Filtering signal D	Kalman filter **D.1**	FIR filter **D.2**	Nothing None **D.3**			
Logic check position E	Sensor data **E.1**					
Logic speed of process F	Constant speed **F.1**	Position dependent**F.2**				
Logic reverse distance G	100% Whole way **G.1**	20% Certain % of motion range **G.2**	STOP Stop, no reverse **G.3**			
Communication with main ECU H	CAN CAN **H.1**	LIN **H.2**				

Concept 1	🟥
Concept 2	🟧
Concept 3	🟩
Concept 4	🟦
Concept 6	🟦
Concept 12	🟪
Concept 16	🟨

Figure 61 The morphological matrix with the paths of combined sub solutions that are used to generate the 7 concepts that remains after the elimination matrix.

Appendix D



COST DESCRIPTION	SEK/HOUR	HOUR		COST
D&D Sunroof Anti-pinch cost SPA1			SEK	15 895 654,40
Development			**SEK**	**10 000 000,00**
Calibration			**SEK**	**4 295 654,40**
Static parametrization			SEK	1 170 374,40
Dinamic parametrization			SEK	2 750 880,00
Climate chamber parametrization	9500	6	SEK	342 000,00
Hällered parametrization	900	6	SEK	32 400,00
Volvo resources administration SPA1			SEK	1 600 000,00
BOM cost Sunroof per ECU (part price)			SEK	8,00
D&D Windows Anti-pinch cost SPA1			SEK	7 962 600,00
Development			**SEK**	**5 000 000,00**
Calibration			**SEK**	**2 562 600,00**
Parametrization complete			SEK	2 188 200,00
Climate chamber parametrization	9500	6	SEK	342 000,00
Hällered parametrization	900	6	SEK	32 400,00
Volvo resources administration SPA1			SEK	400 000,00
BOM cost Window per ECU (part price)			SEK	32,00
D&D Tail gate Anti-pinch cost SPA1			SEK	3 680 000,00
Development			**SEK**	**500 000,00**
Calibration			**SEK**	**2 780 000,00**
Parametrization complete			SEK	500 000,00
Climate chamber parametrization	9500	40	SEK	2 280 000,00
Hällered parametrization	900	0	SEK	-
Volvo resources administration SPA1			SEK	400 000,00
BOM cost Tail gate per ECU (part price)			SEK	8,00

Number of productions sites per carline	2
Number of carline per platform	6
Number of HW variants per carline	2
Number of cars per car line	100000

SPA1 ANTI-PINCH COST DESCRIPTION		COST
D&D	**SEK**	**27 538 254**
Sunroof	SEK	15 895 654
Windows	SEK	7 962 600
Tail gate	SEK	3 680 000
Volvo resources	**SEK**	**2 400 000**
Sunroof	SEK	1 600 000
Windows	SEK	400 000
Tail gate	SEK	400 000
Part price per car	**SEK**	**48**
Sunroof	SEK	8
Windows	SEK	32
Tail gate	SEK	8

VOLVO ANTI-PINCH COST DESCRIPTION		COST
D&D	**SEK**	**2 000 000**
SW development	SEK	1 000 000
Calibration	SEK	1 000 000

Appendix E

Formal Testing Concept 1

Test Results from the formal testing of the sunroof panel and curtain.

Sunroof Panel Position 1						
Measurement	Distance 1: 50[mm]			Distance 2: 100[mm]		
	Pinch Force [N]	Sunroof A [mm]	Sunroof B [mm]	Pinch Force [N]	Sunroof A [mm]	Sunroof B [mm]
1	40	334	335	34	333	332
2	40	248	335	34	250	333
3	35	195	331	33	213	333
4	40	135	335	33	174	333
5	27	87	333	33	140	333
Average	37			34		
Within	14			1		

Sunroof Panel Position 2						
Measurement	Distance 1: 50[mm]			Distance 2: 100[mm]		
	Pinch Force [N]	Sunroof A [mm]	Sunroof B [mm]	Pinch Force [N]	Sunroof A [mm]	Sunroof B [mm]
1	37	330	330	36	327	326
2	37	260	330	32	244	326
3	37	196	330	49	201	325
4	37	129	325	53	170	330
5	37	85	325	48	132	330
Average	37			44		
Within	1			22		

Sunroof Panel Position 3						
Measurement	Distance 1: 50[mm]			Distance 2: 100[mm]		
	Pinch Force [N]	Sunroof A [mm]	Sunroof B [mm]	Pinch Force [N]	Sunroof A [mm]	Sunroof B [mm]
1	43	320	320	56	225	320
2	43	270	325	55	242	321
3	49	196	325	55	183	322
4	45	153	323	55	147	320
5	35	85	325	55	127	321
Average	43			55		
Within	14			1		

4 mm Rod Test			
	P1	P2	P3
	Reacted [Yes/No]	Reacted [Yes/No]	Reacted [Yes/No]
	Yes	Yes	Yes

Sunroof Panel Tilt			
Measurement	P1	P2	P3
	Pinch Force [N]	Pinch Force [N]	Pinch Force [N]
1	47	52	48

4 mm Rod Test			
	P1	P2	P3
	Reacted [Yes/No]	Reacted [Yes/No]	Reacted [Yes/No]
	Yes	Yes	Yes

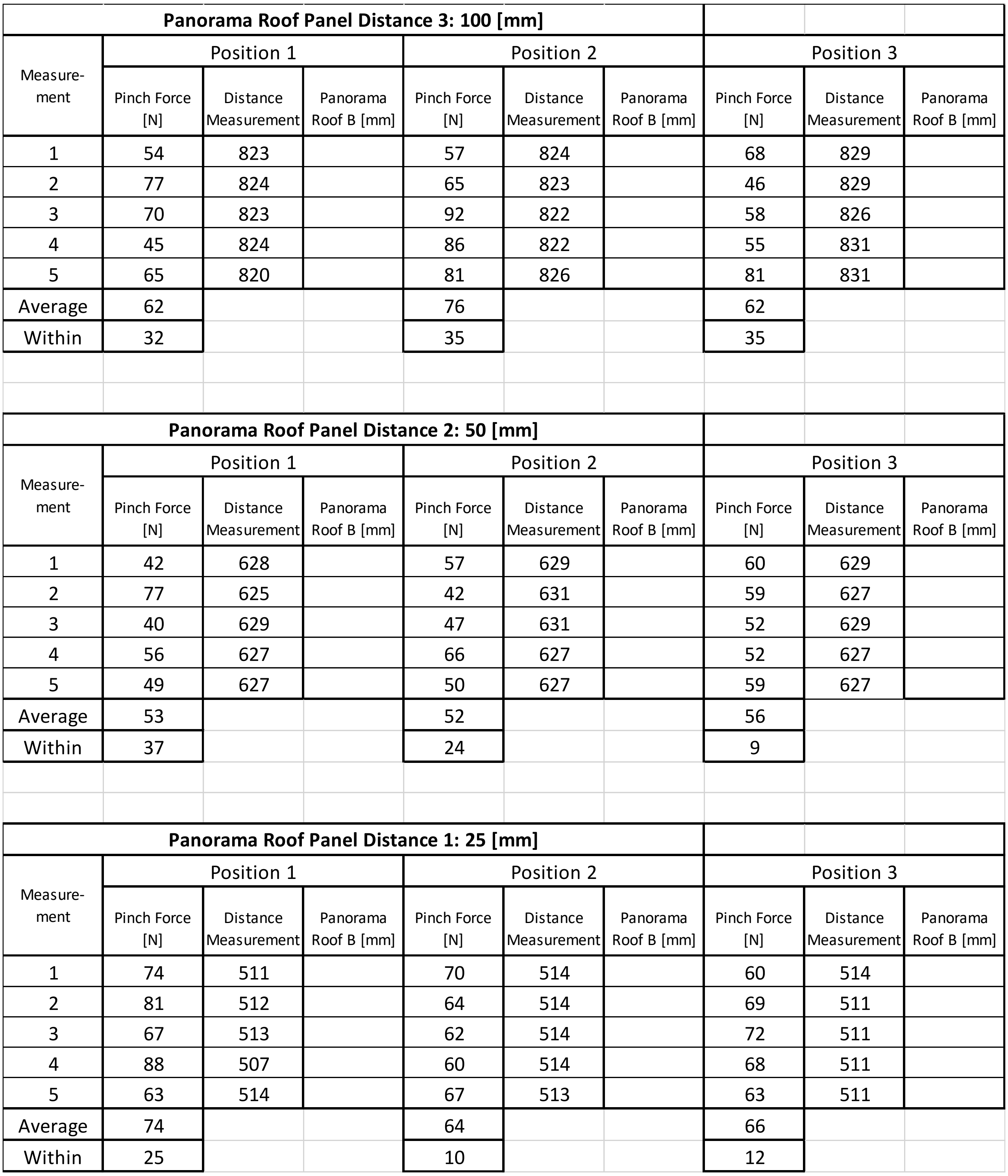
Sunroof Curtain Position 3						
Measurement	Distance 1: 50[mm]			Distance 2: 100[mm]		
	Pinch Force [N]	Sunroof A [mm]	Sunroof B [mm]	Pinch Force [N]	Sunroof A [mm]	Sunroof B [mm]
1	32	694	710	33	710	710
2	34	543	710	35	550	710
3	33	406	710	34	418	710
4	33	280	710	34	288	710
5	33	153	710	34	163	710
Average	33			34		
Within	1			1		

4 mm Rod Test		
P1	P2	P3
Reacted [Yes/No]	Reacted [Yes/No]	Reacted [Yes/No]
Yes	Yes	Yes

Informal Testing Concept 1

Sunroof Panel Concept 1

Test Results from the informal testing of the sunroof panel.



Panorama Roof Panel Distance 3: 100 [mm]									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	54	823		57	824		68	829	
2	77	824		65	823		46	829	
3	70	823		92	822		58	826	
4	45	824		86	822		55	831	
5	65	820		81	826		81	831	
Average	62			76			62		
Within	32			35			35		

Panorama Roof Panel Distance 2: 50 [mm]									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	42	628		57	629		60	629	
2	77	625		42	631		59	627	
3	40	629		47	631		52	629	
4	56	627		66	627		52	627	
5	49	627		50	627		59	627	
Average	53			52			56		
Within	37			24			9		

Panorama Roof Panel Distance 1: 25 [mm]									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	74	511		70	514		60	514	
2	81	512		64	514		69	511	
3	67	513		62	514		72	511	
4	88	507		60	514		68	511	
5	63	514		67	513		63	511	
Average	74			64			66		
Within	25			10			12		

Sunroof Curtain Concept 1

Test Results from the informal testing of the sunroof curtain.

Curtain distance 1 25 mm									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Position 2	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	42	83		25	105		36	95	
2	41	85		37	91		37	95	
3	42	81		31	93		31	97	
4	42	81		28	105		34	97	
5	41	81		31	101		34	97	
Average	42			30			34		
Within	1			12			6		

Panorama Roof Distance 2									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	34	173		44	184		43	145	
2	33	173		39	183		35	147	
3	35	173		39	181		46	143	
4	39	173		43	181		37	145	
5	32	175		42	181		42	145	
Average	35			41			41		
Within	7			5			11		

curtain Roof Distance 3									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	36	317		42	327		32	332	
2	42	319		43	329		38	327	
3	38	313		43	331		37	327	
4	42	317		46	331		38	327	
5	41	313		39	33		39	327	
Average	40			43			37		
Within	6			7			7		

Windows Concept 1

Window distance 1												
Measurement	Position 1			Position 2			Position 3			Position 4		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Position 2	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	58	49		36	56							
2	46	52		32	55							
3	48	49		40	54							
4	55	49		41	49							
5	59	49		33	52							
Average	53			36			########			########		
Within	13			9			0			0		

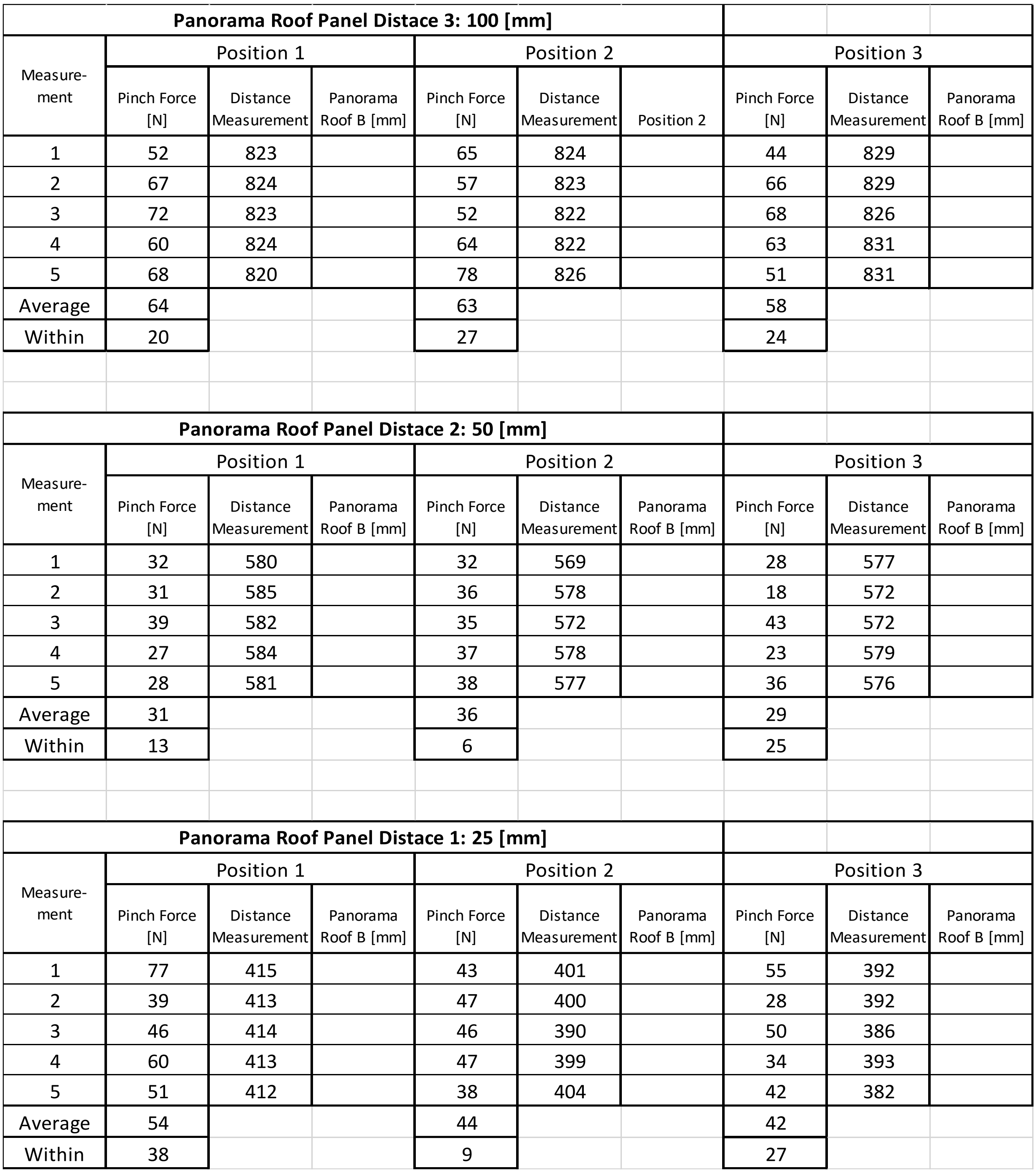
Window distance 2												
Measurement	Position 1			Position 2			Position 3			Position 4		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	53	92										
2	60	91										
3	52	91										
4	46	92										
5	56	90										
Average	53			########			########			########		
Within	14			0			0			0		

Window distance 3												
Measurement	Position 1			Position 2			Position 3			Position 4		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	52	151										
2	45	151										
3	54	151										
4	55	150										
5	52	149										
Average	52			########			########			########		
Within	10			0			0			0		

Informal Testing Concept 2

Sunroof Panel Concept 2

Test Results from the informal testing of the sunroof panel.



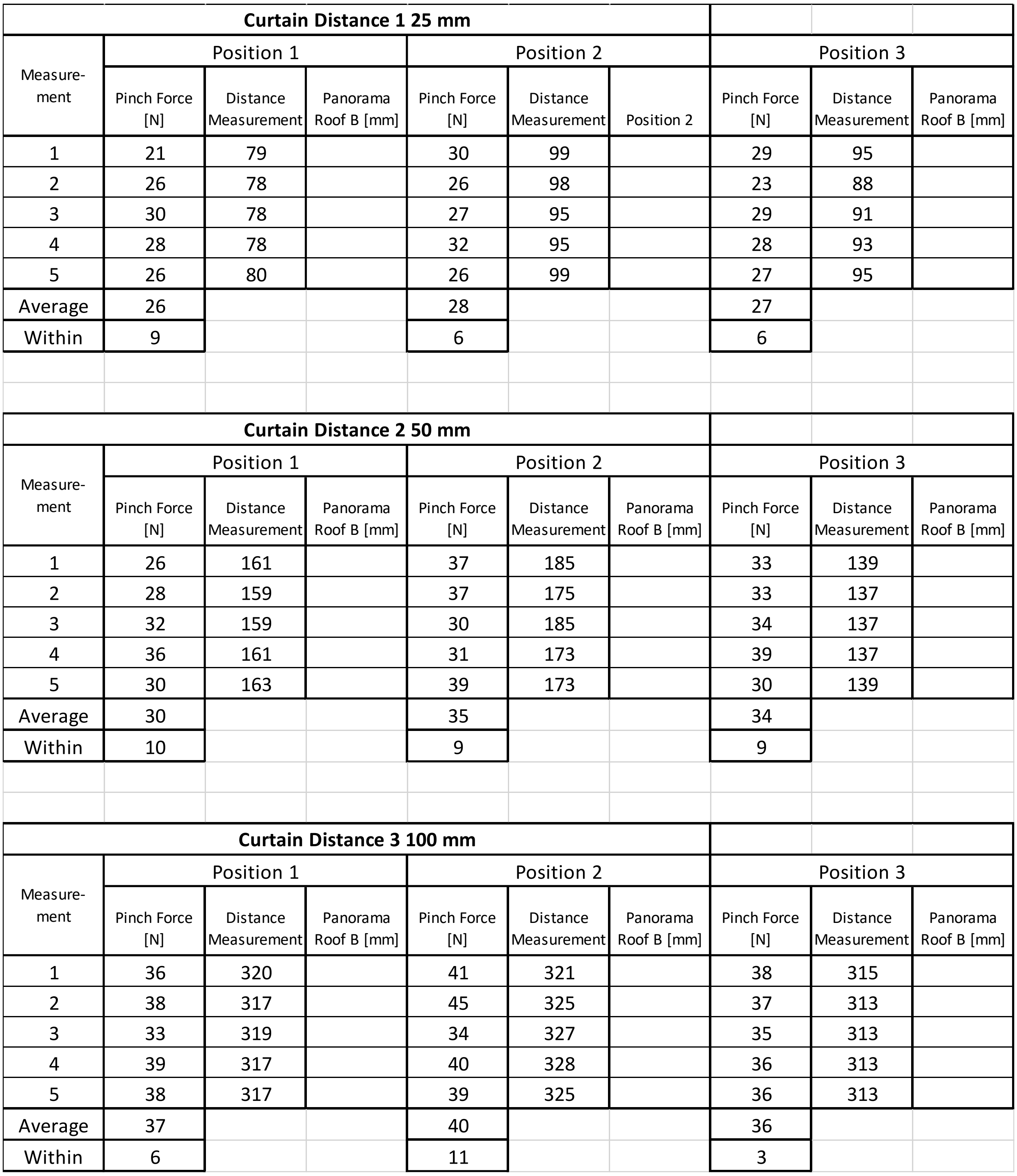
Panorama Roof Panel Distace 3: 100 [mm]									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Position 2	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	52	823		65	824		44	829	
2	67	824		57	823		66	829	
3	72	823		52	822		68	826	
4	60	824		64	822		63	831	
5	68	820		78	826		51	831	
Average	64			63			58		
Within	20			27			24		

Panorama Roof Panel Distace 2: 50 [mm]									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	32	580		32	569		28	577	
2	31	585		36	578		18	572	
3	39	582		35	572		43	572	
4	27	584		37	578		23	579	
5	28	581		38	577		36	576	
Average	31			36			29		
Within	13			6			25		

Panorama Roof Panel Distace 1: 25 [mm]									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	77	415		43	401		55	392	
2	39	413		47	400		28	392	
3	46	414		46	390		50	386	
4	60	413		47	399		34	393	
5	51	412		38	404		42	382	
Average	54			44			42		
Within	38			9			27		

Sunroof Curtain Concept 2

Test Results from the informal testing of the sunroof curtain.



Curtain Distance 1 25 mm									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Position 2	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	21	79		30	99		29	95	
2	26	78		26	98		23	88	
3	30	78		27	95		29	91	
4	28	78		32	95		28	93	
5	26	80		26	99		27	95	
Average	26			28			27		
Within	9			6			6		

Curtain Distance 2 50 mm									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	26	161		37	185		33	139	
2	28	159		37	175		33	137	
3	32	159		30	185		34	137	
4	36	161		31	173		39	137	
5	30	163		39	173		30	139	
Average	30			35			34		
Within	10			9			9		

Curtain Distance 3 100 mm									
Measure-ment	Position 1			Position 2			Position 3		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	36	320		41	321		38	315	
2	38	317		45	325		37	313	
3	33	319		34	327		35	313	
4	39	317		40	328		36	313	
5	38	317		39	325		36	313	
Average	37			40			36		
Within	6			11			3		

Windows Concept 1

Window distance 1 25 mm												
Measure-ment	Position 1			Position 2			Position 3			Position 4		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Position 2	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	39	58		28	45							
2	32	64		25	50							
3	38	61		25	70							
4	40	62		19	76							
5	31	45		25	70							
Average	36			24			########			########		
Within	9			9			0			0		

Window distance 50 mm												
Measure-ment	Position 1			Position 2			Position 3			Position 4		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	41	105										
2	40	105										
3	46	105										
4	48	96										
5	54	98										
Average	46			########			########			########		
Within	14			0			0			0		

Window distance 3												
Measure-ment	Position 1			Position 2			Position 3			Position 4		
	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]	Pinch Force [N]	Distance Measurement	Panorama Roof B [mm]
1	73	179										
2	71	177										
3	60	172										
4	64	172										
5	70	176										
Average	68			########			########			########		
Within	13			0			0			0		

Appendix F

Arduino code concept 1.

```
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This proram uses the hall sensor input from the panel, curtain and front window motors to determine their position and speed //
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////// Initiate Parameters Panel ///////////////////////////////////////////////////

volatile byte hsDetectPanel = 0;          // Hall Sensor detection in the panel
const byte hSensitivityPanel = 1;         // How sensitive will the system be / the readings of the hall sensors
volatile byte confirmerPanel = 0;         // Vairable used to help determine the distance

unsigned long hs1TimePanel = 0;           // Time when hall sensor 1 is detected
unsigned long hs2TimePanel = 0;           // Time when hall sensor 2 is detected

// Parameters for determining if the panel has reached its end position
long progTimePanel = 0;                   // A timer to start timing when the user presses the button
long fullyOpenedTimerPanel = 0;           // A timer to measure time between the last hall sensor input and program time while closing the panel
long fullyClosedTimerPanel = 0;           // A timer to measure time between the last hall sensor input and program time while opening the panel
byte oneTimeCheckPanel = 0;               // A flag to initiate values of fullyOpened and fullyClosed timers
byte fullyOpenPanel = 0;                  // A flag that tells the panel is in fully opened position
byte fullyClosedPanel = 0;                // A flag that tells the panel is in fully closed position
byte bufferOpenPanel = 0;                 // A one time buffer that allows the panel to open for one more program cycle when panel movement is stopped either by user or hard stop
byte bufferClosePanel = 0;                // A one time buffer that allows the panel to close for one more program cycle when panel movement is stopped either by user or hard stop
const int hardStopTimePanel = 100;        // Time in milliseconds for hard stop detection to trigger
const int fullyOpenDistancePanel = 1390;  // The distance value that is forcefully assigned when panel is fully open
const int fullyClosedDistancePanel = 0;   // The distance value that is forcefully assigned when panel is fully closed

long Time_Between_Hs_Input_Panel = 0;     // The time between Hall Sensor inputs, gives information about how fast the panel is moving / it is the reverese of rpm
long Time_Between_Hs_Input_Panel_Old = 0; // Time between Hall Sensor Inputs in earlier iteration
long Time_Per_Revolution_Panel = 0;       // The time it takes to get 4 Hs inputs --> for half a rotation

int distancePanel = 0;                    // The number of Hs inputs recived from the starting position of the panel (fully closed)
long Tresh_base_pos = 0;                  // Parameter that enables the curve to adapt to the characteristics of the current operation
const int pinchSensitivityPanel = 0;      // Parameter that enables change of the amplitude of the treshold curve for all zones

 //Pinching Zones upper & lowe limits
const int UL1Panel = 1050;                // Upper limit of the distance for Pinch Zone 1
const int LL1Panel = 650;                 // Lower limit of the distance for Pinch Zone 1

const int UL2Panel = LL1Panel;            // Upper limit of the distance for Pinch Zone 2
const int LL2Panel = 520;                 // Lower limit of the distance for Pinch Zone 2

const int UL3Panel = LL2Panel;            // Upper limit of the distance for Pinch Zone 3
const int LL3Panel = 380;                 // Lower limit of the distance for Pinch Zone 3

const int UL4Panel = LL3Panel;            // Upper limit of the distance for Pinch Zone 4
const int LL4Panel = 230;                 // Lower limit of the distance for Pinch Zone 4

const int UL5Panel = LL4Panel;            // Upper limit of the distance for Pinch Zone 5
const int LL5Panel = 140;                 // Lower limit of the distance for Pinch Zone 5

const int UL6Panel = LL5Panel;            // Upper limit of the distance for Pinch Zone 6
const int LL6Panel = 90;                  // Lower limit of the distance for Pinch Zone 6

const int UL7Panel = LL6Panel;            // Upper limit of the distance for Pinch Zone 7
const int LL7Panel = 60;                  // Lower limit of the distance for Pinch Zone 7

// Treshold curve fit parameters on the form:  A* Distance^2 + B* Distance + C for the Panel
// Zone 1
const float A1Panel = -0.0015;
const float B1Panel = 0.0318;
int C1Panel_tresh    = 170;               // C value for the adaptive Curve
int C1Panel          = 4150;

// Zone 2
const float A2Panel = 0.0052;
const float B2Panel = 0.3854;
int C2Panel_tresh    = 0;                 // C value for the adaptive Curve
int C2Panel          =4000;

// Zone 3
const float A3Panel = -0.04;
const float B3Panel = 6;
int C3Panel_tresh    = 140;               // C value for the adaptive Curve
int C3Panel          = 4150;

// Zone 4
const float A4Panel = 0.029;
const float B4Panel = -3.31;
int C4Panel_tresh    = 150;               // C value for the adaptive Curve
int C4Panel          = 4150;

// Zone 5
const float A5Panel = 0.11;
const float B5Panel =-2.5;
int C5Panel_tresh   = 150;               // C value for the adaptive Curve
int C5Panel         = 4150;

// Zone 6
const float A6Panel = 0;
const float B6Panel = 0;
int C6Panel_tresh   = 1150;              // C value for the adaptive Curve
int C6Panel         = 5200;

// Zone 7
const float A7Panel = 0;
const float B7Panel = -5.4011;
int C7Panel_tresh   =  1150;             // C value for the adaptive Curve
int C7Panel         = 0;


// Parameters used to set a state, varies between 1 & 0 depending on if they are on or off
int pinchDetectorPanel = 0;              // Sets the system in "Pinch mode" 0 = Nothing is happening 1 = A pinch has occured

int stopInPinchZonePanel = 0;            // Allows the system to overwrite a Pinch if the user stops the pannel within the "Pinch Zon
e" avoids fals pinches 0 = Nothing is happening 1 = Panel has stoped in the pinch zone
int stopDistancePanel = 0;               // Sets the distance at which the panel has been stoped

//////////////////////////////////////////////////// Initiate Parameters Curtain /////////////////////////////////////////////////

volatile byte hsDetectCurtain  = 0;      // Hall Sensor detection in the Curtain
const byte hSensitivityCurtain = 1;      // How sensitive will the system be / the readings of the hall sensors
volatile byte confirmerCurtain = 0;      // Vairable used to help determine the distance

unsigned long hs1CurtainTime   = 0;      // Time when hall sensor 1 is detected
unsigned long hs2CurtainTime   = 0;      // Time when hall sensor 2 is detected



// Parameters for determining if the Curtain has reached its end position
long progTimeCurtain         = 0;             // A timer to start timing when the user presses the button
long fullyOpenedTimerCurtain = 0;             // A timer to measure time between the last hall sensor input and program time while cl
osing the curtain
long fullyClosedTimerCurtain = 0;             // A timer to measure time between the last hall sensor input and program time while op
ening the curtain
byte oneTimeCheckCurtain     = 0;             // A flag to initiate values of fullyOpened and fullyClosed timers
byte fullyOpenCurtain        = 0;             // A flag that tells the curtain is in fully opened position
byte fullyClosedCurtain      = 0;             // A flag that tells the curtain is in fully closed position
byte bufferOpenCurtain       = 0;             // A one time buffer that allows the curtain to open for one more program cycle when cu
rtain movement is stopped either by user or hard stop
byte bufferCloseCurtain      = 0;             // A one time buffer that allows the curtain to close for one more program cycle when c
urtain movement is stopped either by user or hard stop
const int hardStopTimeCurtain        = 100;   // Time in milliseconds for hard stop detection to trigger
const int fullyOpenDistanceCurtain   = 2160;  // The distance value that is forcefully assigned when curtain is fully open
const int fullyClosedDistanceCurtain = 0;     // The distance value that is forcefully assigned when curtain is fully closed

long Time_Between_Hs_Input_Curtain    = 0;    // The time between Hall Sensor inputs, gives information about how fast the Curtain is
 moving / it is the reverese of rpm
long Time_Between_Hs_Input_CurtainOld = 0;    // Time between Hall Sensor Inputs in earlier iteration
long Time_Per_Revolution_Curtain      = 0;    // The time it takes to get 4 Hs inputs --> for half a revolution

int distanceCurtain         = 0;              // The number of Hs inputs recived from the starting position of the Curtain (fully clo
sed)
int SensitivityCurtain      = 0;              // Parameter that enables change of the amplitude of the treshold curve for all zones
long Tresh_base_Curtain     = 0;              // Parameter that enables the curve to adapt to the characteristics of the current oper
ation

// Upper & lowe limits for the Pinch Zones
const int UL1Curtain = 956;                   // Upper limit of the distance for Pinch Zone 1
const int LL1Curtain = 540;                   // Lower limit of the distance for Pinch Zone 1

const int UL2Curtain = LL1Curtain;            // Upper limit of the distance for Pinch Zone 2
const int LL2Curtain = 210;                   // Lower limit of the distance for Pinch Zone 2

const int UL3Curtain = LL2Curtain;            // Upper limit of the distance for Pinch Zone 3
const int LL3Curtain = 100;                   // Lower limit of the distance for Pinch Zone 3

const int UL4Curtain = LL3Curtain;            // Upper limit of the distance for Pinch Zone 4
const int LL4Curtain = 20;                    // Lower limit of the distance for Pinch Zone 4

// Treshold curve fit parameters on the form:  A* Distance^2 + B* Distance + C. Straight line for curtain so C is tha amplitude
// Zone 1
const int C1Curtain_tresh = 200;              // C value for the adaptive Curve
int C1Curtain             = 5000;

// Zone 2
const int C2Curtain_tresh = 300;              // C value for the adaptive Curve
int C2Curtain             = 5100;

// Zone 3
const int C3Curtain_tresh = 400;              // C value for the adaptive Curve
int C3Curtain             = 5200;

// Zone 4
const int C4Curtain_tresh = 500;              // C value for the adaptive Curve
int C4Curtain             = 5300;

// Parameters used to set a state, varies between 1 & 0 depending on if they are on or off
int pinchDetectorCurtain   = 0;               // Sets the system in "Pinch mode" 0 = Nothing is happening 1 = A pinch has occured

int stopInPinchZoneCurtain = 0;               // Allows the system to overwrite a Pinch if the user stops the Curtain within the "Pin
ch Zone" avoids fals pinches 0 = Nothing is happening 1 = Curtain has stoped in the pinch zone
int stopDistanceCurtain    = 0;               // Sets the distance at which the Curtain has been stoped
```

```
///////////////////////////////////////////////// Initiate Parameters Window ///////////////////////////////////////////////////

volatile byte hsDetectWindow  = 0;            // Hall Sensor detection in the Window
volatile byte confirmerWindow = 0;            // Vairable used to help determine the distance
const byte hSensitivityWindow = 1;            // How sensitive will the system be / the readings of the hall sensors

unsigned long hs1TimeWindow = 0;              // Time when hall sensor 1 is detected
unsigned long hs2TimeWindow = 0;              // Time when hall sensor 2 is detected

// Parameters for determining if the Window has reached its end position
long progTimeWindow         = 0;              // A timer to start timing when the user presses the button
long fullyOpenedTimerWindow = 0;              // A timer to measure time between the last hall sensor input and program time while cl
osing the panel
long fullyClosedTimerWindow = 0;              // A timer to measure time between the last hall sensor input and program time while op
ening the panel
byte oneTimeCheckWindow     = 0;              // A flag to initiate values of fullyOpened and fullyClosed timers
byte fullyOpenWindow        = 0;              // A flag that tells the panel is in fully opened position
byte fullyClosedWindow      = 0;              // A flag that tells the panel is in fully closed position
byte bufferOpenWindow       = 0;              // A one time buffer that allows the panel to open for one more program cycle when pane
l movement is stopped either by user or hard stop
byte bufferCloseWindow      = 0;              // A one time buffer that allows the panel to close for one more program cycle when pan
el movement is stopped either by user or hard stop
const int hardStopTimeWindow        = 120;    // Time in milliseconds for hard stop detection to trigger
const int fullyOpenDistanceWindow   = 460;    // The distance value that is forcefully assigned when panel is fully open
const int fullyClosedDistanceWindow = 0;      // The distance value that is forcefully assigned when panel is fully closed



long Time_Between_Hs_Input_Window    = 0;     // The time between Hall Sensor inputs, gives information about how fast the Window is
moving / it is the reverese of rpm
long Time_Between_Hs_Input_WindowOld = 0;     // Time between Hall Sensor Inputs in earlier iteration
long Time_Per_Revolution_Window      = 0;     // The time it takes to get 4 Hs inputs --> for half a revolution

int distanceWindow                   = 0;     // The number of Hs inputs recived from the starting position of the Window (fully clos
ed)
long Tresh_base_Window               = 0;     // Parameter that enables the curve to adapt to the characteristics of the current oper
ation
int sensitivityWindow                = 0;     // Parameter that enables change of the amplitude of the treshold curve for all zones

// Treshold section parameters
const int UL1Window = 310;                    // Upper limit of the distance for Pinch Zone 1
const int LL1Window = 270;                    // Lower limit of the distance for Pinch Zone 1

const int UL2Window = LL1Window;              // Upper limit of the distance for Pinch Zone 2
const int LL2Window = 235;                    // Lower limit of the distance for Pinch Zone 2

const int UL3Window = LL2Window;              // Upper limit of the distance for Pinch Zone 3
const int LL3Window = 145;                    // Lower limit of the distance for Pinch Zone 3

const int UL4Window = LL3Window;              // Upper limit of the distance for Pinch Zone 4
const int LL4Window = 85;                    // Lower limit of the distance for Pinch Zone 4

const int UL5Window = LL4Window;              // Upper limit of the distance for Pinch Zone 5
const int LL5Window = 50;                     // Lower limit of the distance for Pinch Zone 5

const int UL6Window = LL5Window;              // Upper limit of the distance for Pinch Zone 6
const int LL6Window = 30;                     // Lower limit of the distance for Pinch Zone 6

// Treshold curve fit parameters on the form:  A* Distance^2 + B* Distance + C for the different zones
// Values comes from Window
const float A1Window = 0;
const float B1Window = 20;
int C1Window         = 10270;
const int C1Window_thresh = 700;              // C value for the adaptive Curve

const float A2Window = 0;
const float B2Window = -5;
int C2Window         = 11270;
const int C2Window_thresh = 1500;             // C value for the adaptive Curve

const float A3Window = 0;
const float B3Window = -5;
int C3Window         = 10770;
const int C3Window_thresh = 1200;             // C value for the adaptive Curve

const float A4Window = 0;
const float B4Window = 5;
int C4Window         = 10460;
const int C4Window_thresh = 800;             // C value for the adaptive Curve

const float A5Window = 0;
const float B5Window = 2;
int C5Window         = 11000;
const int C5Window_thresh = 1000;              // C value for the adaptive Curve

const float A6Window = 0;
const float B6Window = 2;
int C6Window         = 10000;
const int C6Window_thresh = 1200;              // C value for the adaptive Curve

const int pinchSensitivityWindow = 0;         // Used to lower or raise the curve against the pace input to change the sensitivity of
 the treshold curve fit

// Parameters used to set a state, varies between 1 & 0 depending on if they are on or off
int pinchDetectorWindow   = 0;                // Sets the system in "Pinch mode" 0 = Nothing is happening 1 = A pinch has occured
int stopInPinchZoneWindow = 0;                // Allows the system to overwrite a Pinch if the user stops the window within the "Pinc
h Zone" avoids fals pinches 0 = Nothing is happening 1 = Window has stoped in the pinch zone
int stopDistanceWindow    = 0;                // Sets the distance at which the Window has been stoped

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// Defining input/output pins

////////////////////////// Input/Output pins Panel
const int closeButtonPanel   = 6;             // Defines the pin on the Arduino for the button that closes the panel
const int openButtonPanel    = 7;             // Defines the pin on the Arduino for the button that opens the panel
const int hs1Panel           = 20;            // Defines the pin on the Arduino which recives the input from hall sensor 1 for the pa
nel
const int hs2Panel           = 21;            // Defines the pin on the Arduino which recives the input from hall sensor 2 for the pa
nel

//////////////////////// Input/Output pins Curtain
const int closeButtonCurtain = 4;             // Defines the pin on the Arduino for the button that closes the curtain
const int openButtonCurtain  = 5;             // Defines the pin on the Arduino for the button that opens the curtain
const int hs1Curtain         = 18;            // Defines the pin on the Arduino which recives the input from hall sensor 1 for the cu
rtain
const int hs2Curtain         = 19;            // Defines the pin on the Arduino which recives the input from hall sensor 2 for the cu
rtain

////////////////////////// Input/Output pins Window
const int closeButtonWindow  = 4;             // Defines the pin on the Arduino for the button that closes the window
const int openButtonWindow   = 5;             // Defines the pin on the Arduino for the button that opens the window
const int hs1Window          = 2;             // Defines the pin on the Arduino which recives the input from hall sensor 1 for the wi
ndow (only one hs for window)
const byte curtainVSwindow   = 3;             // Defines the pin on the Arduino which says if the switch is active for the window or
the curtain

/////////////////////// Controlling the direktion & speed (allways 100%/255 for this concept)

const byte PHpinPanel = 9;                    // Corresponds to direction of rotation (OCR2B)
const byte ENpinPanel = 10;                   // Corresponds to PulsWdtMod duty cycle (OCR2A)
const byte PHpinCurtain = 11;                 // Corresponds to direction of rotation (OCR1A)
const byte ENpinCurtain = 12;                 // COrresponds to PulsWdtMod duty cycle (OCR1B)

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////// All parameters and variables defined /////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup() // Void Setup
{
  Serial.begin(115200);

///////////////////////////// Setup direction & Speed Control

// PWM control changing frequency for TIMER 1
  TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM20);
  TCCR1B = TCCR1B & 0b11111000 | 0x01;

// PWM control changing frequency for TIMER 2
  TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM20);
  TCCR2B = TCCR2B & 0b11111000 | 0x01;



// Setting the pin mode to HIGH for PH/EN pins for
  pinMode(PHpinPanel, OUTPUT);
  pinMode(ENpinPanel, OUTPUT);
  pinMode(PHpinCurtain, OUTPUT);
  pinMode(ENpinCurtain, OUTPUT);

  ////////////////////////// Void Setup Panel

  pinMode(openButtonPanel, INPUT);        // Open Button is an input
  pinMode(closeButtonPanel, INPUT);       // Close Button is an input
  OCR2A = 0;                              // Initial state of Direction & speed is 0 = nothing happens before button is pressed
  OCR2B = 0;                              // Initial state of Direction & speed is 0 = nothing happens before button is pressed

  attachInterrupt(digitalPinToInterrupt(hs1Panel), magnet_detect1Panel, RISING); // Interuppts the system if a hall sensor is detecte
d
  attachInterrupt(digitalPinToInterrupt(hs2Panel), magnet_detect2Panel, RISING); // Interuppts the system if a hall sensor is detecte
d

  ////////////////////////// Setup Curtain

  pinMode(openButtonCurtain, INPUT);      // Open Button is an input
  pinMode(closeButtonCurtain, INPUT);     // Close Button is an input
  OCR1A = 0;                              // Initial state of Direction & speed is 0 = nothing happens before button is pressed
  OCR1B = 0;                              // Initial state of Direction & speed is 0 = nothing happens before button is pressed

  attachInterrupt(digitalPinToInterrupt(hs1Curtain), magnet_detect1Curtain, RISING); // Interuppts the system if a hall sensor is det
ected
  attachInterrupt(digitalPinToInterrupt(hs2Curtain), magnet_detect2Curtain, RISING); // Interuppts the system if a hall sensor is det
ected

  ////////////////////////// Setup Window

  pinMode(openButtonWindow, INPUT);       // Open Button is an input
  pinMode(closeButtonWindow, INPUT);      // Close Button is an input
  pinMode(curtainVSwindow, INPUT);
  attachInterrupt(digitalPinToInterrupt(hs1Window), magnet_detect1Window, RISING); // Interuppts the system if a hall sensor is detec
ted (only one for window)

} // Ends the void setup
```

```
void loop()
{
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// PANEL
////////////////////////// OPEN Panel

// Enter the open statement if the open buttons is presses and the panel is not in its fully open position
  if (digitalRead(openButtonPanel) == HIGH && fullyOpenPanel == 0)
  {
    fullyClosedPanel = 0;               // Resets the "fullyClosedPanel" flag to 0
    stopInPinchZonePanel = 0;           // Resets the "stopInPinchZonePanel" flag to 0

    OCR2A = 255;                        // Speed = 100%
    OCR2B = 0;                          // Opening direction
    DistanceOpenFunkPanel();            // Calls the Distance open function
  }                                     // Ends the opening statement

  ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
  // CLOSE Panel

// Enter the close statement if the close buttons is presses, the panel is not in its fully closed position,
//  if there is no stop in the pinch zone, and if no pinch is detected

  else if (digitalRead(closeButtonPanel) == HIGH && stopInPinchZonePanel == 0 && pinchDetectorPanel == 0 && fullyClosedPanel == 0)
  {
    fullyOpenPanel = 0;                 // Resets the "fullyOpenPanel" flag to 0
    OCR2A = 255;                        // Speed = 100%
    OCR2B = 255;                        // Closing direction

    if (pinchDetectorPanel == 0)        // enters if no pinch is detected
    {
      DistanceCloseFunkPanel();         // Calls the Distance close function
    }

    if (distancePanel > UL1Panel && distancePanel <= UL1Panel + 4) // Adaptive threshold curve
    {

      if (Time_Per_Revolution_Panel > Tresh_base_pos)              // If The time per revolution is higher than the treshold base wil
l a new treshold value be given
      {
        Tresh_base_pos = Time_Per_Revolution_Panel;                // A new base value is assigned to the treshholdcurve that is uniq
ue for this operation
      }

      // Asigned the Adaptive treshold base value to the C values for the curve
      C1Panel = Tresh_base_pos + C1Panel_tresh;           // Zone 1
      C2Panel = Tresh_base_pos + C2Panel_tresh;           // Zone 2
      C3Panel = Tresh_base_pos + C3Panel_tresh;           // Zone 3
      C4Panel = Tresh_base_pos + C4Panel_tresh;           // Zone 4
      C5Panel = Tresh_base_pos + C5Panel_tresh;           // Zone 5
      C6Panel = Tresh_base_pos + C6Panel_tresh;           // Zone 6
      C7Panel = Tresh_base_pos + C7Panel_tresh;           // Zone 7
    }

    else if (distancePanel >= LL1Panel && distancePanel < UL1Panel)     // Enters Pinch Zone 1
    {

      // Checks if the treshold curve for Zone 1 is lower than the "time per revolution"
      if (A1Panel * (UL1Panel - distancePanel) * (UL1Panel - distancePanel) + B1Panel * (UL1Panel - distancePanel) + C1Panel - pinchS
ensitivityPanel < Time_Per_Revolution_Panel)
      {
                pinchDetectorPanel = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

                // Printind details about the pinch situation
                Serial.print(round(A1Panel * (UL1Panel - distancePanel) * (UL1Panel - distancePanel) + B1Panel * (UL1Panel - distance
Panel) + C1Panel - pinchSensitivityPanel));
                Serial.print("; ");
                Serial.print(distancePanel);
                Serial.print("; ");
                Serial.print(Time_Per_Revolution_Panel);
                Serial.print("; ");
                Serial.print("Zone1");
                Serial.print("; ");
                Serial.println(Tresh_base_pos);
      }
    }
    else if (distancePanel >= LL2Panel && distancePanel < UL2Panel)  // Enters Pinch Zone 2
    {
      // Checks if the treshold curve for Zone 2 is lower than the "time per revolution"
      if (A2Panel * (UL2Panel - distancePanel) * (UL2Panel - distancePanel) + B2Panel * (UL2Panel - distancePanel) + C2Panel - pinchS
ensitivityPanel < Time_Per_Revolution_Panel)
      {
                pinchDetectorPanel = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

                // Printind details about the pinch situation
                Serial.print(round(A2Panel * (UL2Panel - distancePanel) * (UL2Panel - distancePanel) + B2Panel * (UL2Panel - distance
Panel) + C2Panel - pinchSensitivityPanel));
                Serial.print("; ");
                Serial.print(distancePanel);
                Serial.print("; ");
                Serial.print(Time_Per_Revolution_Panel);
                Serial.print("; ");
                Serial.print("Zone2");
                Serial.print("; ");
                Serial.println(Tresh_base_pos);
      }
    }
    else if (distancePanel >= LL3Panel && distancePanel < UL3Panel) // Enters Pinch Zone 3
    {
      // Checks if the treshold curve for Zone 3 is lower than the "time per revolution"
      if (A3Panel * (UL3Panel - distancePanel) * (UL3Panel - distancePanel) + B3Panel * (UL3Panel - distancePanel) + C3Panel - pinchSensitivityPanel < Time_Per_Revolution_Panel)
      {
                pinchDetectorPanel = 1;       // If the treshold curve is lower than "time per revolution" is a pinch detected --> pinchdetector = 1

                // Printind details about the pinch situation
                Serial.print(round(A3Panel * (UL3Panel - distancePanel) * (UL3Panel - distancePanel) + B3Panel * (UL3Panel - distancePanel) + C3Panel - pinchSensitivityPanel));
                Serial.print("; ");
                Serial.print(distancePanel);
                Serial.print("; ");
                Serial.print(Time_Per_Revolution_Panel);
                Serial.print("; ");
                Serial.print("Zone3");
                Serial.print("; ");
                Serial.println(Tresh_base_pos);
      }
    }
    else if (distancePanel >= LL4Panel && distancePanel < UL4Panel) // Enters Pinch Zone 4
    {
      // Checks if the treshold curve for Zone 4 is lower than the "time per revolution"
      if (A4Panel * (UL4Panel - distancePanel) * (UL4Panel - distancePanel) + B4Panel * (UL4Panel - distancePanel) + C4Panel - pinchSensitivityPanel < Time_Per_Revolution_Panel)
      {
                pinchDetectorPanel = 1;  // If the treshold curve is lower than "time per revolution" is a pinch detected --> pinchdetector = 1

                // Printind details about the pinch situation
                Serial.print(round(A4Panel * (UL4Panel - distancePanel) * (UL4Panel - distancePanel) + B4Panel * (UL4Panel - distancePanel) + C4Panel - pinchSensitivityPanel));
                Serial.print("; ");
                Serial.print(distancePanel);
                Serial.print("; ");
                Serial.print(Time_Per_Revolution_Panel);
                Serial.print("; ");
                Serial.print("Zone4");
                Serial.print("; ");
                Serial.println(Tresh_base_pos);

      }
    }
    else if (distancePanel >= LL5Panel && distancePanel < UL5Panel)  // Enters Pinch Zone 5
    {
      // Checks if the treshold curve for Zone 5 is lower than the "time per revolution"
      if (A5Panel * (UL5Panel - distancePanel) * (UL5Panel - distancePanel) + B5Panel * (UL5Panel - distancePanel) + C5Panel - pinchSensitivityPanel < Time_Per_Revolution_Panel)
      {
                pinchDetectorPanel = 1;    // If the treshold curve is lower than "time per revolution" is a pinch detected --> pinchdetector = 1

                // Printind details about the pinch situation
                Serial.print(round(A5Panel * (UL5Panel - distancePanel) * (UL5Panel - distancePanel) + B5Panel * (UL5Panel - distancePanel) + C5Panel - pinchSensitivityPanel));
                Serial.print("; ");
                Serial.print(distancePanel);
                Serial.print("; ");
                Serial.print(Time_Per_Revolution_Panel);
                Serial.print("; ");
                Serial.print("Zone5");
                Serial.print("; ");
                Serial.println(Tresh_base_pos);

      }
    }
    else if (distancePanel >= LL6Panel && distancePanel < UL6Panel)       // Enters Pinch Zone 6
    {
      // Checks if the treshold curve for Zone 6 is lower than the "time per revolution"
      if (A6Panel * (UL6Panel - distancePanel) * (UL6Panel - distancePanel) + B6Panel * (UL6Panel - distancePanel) + C6Panel - pinchSensitivityPanel < Time_Per_Revolution_Panel)
      {
                pinchDetectorPanel = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --> pinchdetector = 1

                // Printind details about the pinch situation
                Serial.print(round(A6Panel * (UL6Panel - distancePanel) * (UL6Panel - distancePanel) + B6Panel * (UL6Panel - distancePanel) + C6Panel - pinchSensitivityPanel));
                Serial.print("; ");
                Serial.print(distancePanel);
                Serial.print("; ");
                Serial.print(Time_Per_Revolution_Panel);
                Serial.print("; ");
                Serial.print("Zone6");
                Serial.print("; ");
                Serial.println(Tresh_base_pos);
      }
    }
    else if (distancePanel >= LL7Panel && distancePanel < UL7Panel)     // Enters Pinch Zone 7
    {
      // Checks if the treshold curve for Zone 7 is lower than the "time per revolution"
      if (A7Panel * (UL7Panel - distancePanel) * (UL7Panel - distancePanel) + B7Panel * (UL7Panel - distancePanel) + C7Panel - pinchS
ensitivityPanel < Time_Per_Revolution_Panel)
      {

                pinchDetectorPanel = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

                // Printind details about the pinch situation
                Serial.print(round(A7Panel * (UL7Panel - distancePanel) * (UL7Panel - distancePanel) + B7Panel * (UL7Panel - distance
Panel) + C7Panel - pinchSensitivityPanel));
                Serial.print("; ");
                Serial.print(distancePanel);
                Serial.print("; ");
                Serial.print(Time_Per_Revolution_Panel);
                Serial.print("; ");
                Serial.print("Zone7");
                Serial.print("; ");
                Serial.println(Tresh_base_pos);
      }
    }
  } //          Stops Close statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

  else if (pinchDetectorPanel == 1) // If a pinch is detected in the pannel will it change direction and open fully
  {
    OCR2A = 255;                        // Speed 100%
    OCR2B = 0;                          // Opening direction
    DistanceOpenFunkPanel();            // Cals the Distance opening function
  }                                     // Stops pinch in panel statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///// Avoids falls pinches by overwriting the pinch stement if the user stops the panel within the pinch zone
// Enters if Close button is high, stop in pinch zone is high and the pannel is not fully closed

  else if (digitalRead(closeButtonPanel) == HIGH && stopInPinchZonePanel == 1 && fullyClosedPanel == 0)
  {

    OCR2A = 255;                        // Speed 100%
    OCR2B = 255;                        // Closing direction
    DistanceCloseFunkPanel();           // Cals the Distance closing function
  }                                     // Stops Stop in pinch zone statement

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////// Allow the user to stop the panel within the pinch zone
// Enters if both buttons are low and the panel is in the pinch zone
  else if (digitalRead(openButtonPanel) == LOW && digitalRead(closeButtonPanel) == LOW && distancePanel > LL7Panel && distancePanel <
 UL1Panel)
  {
    fullyOpenPanel = 0;                 // Resets the "fullyOpenPanel" flag to 0
    fullyClosedPanel = 0;               // Resets the "fullyClosedPanel" flag to 0

    OCR2A = 0;                          // Speed 0%
    OCR2B = 0;                          // Opening direction/ doesn't matter which direction

    stopInPinchZonePanel = 1;           // Sets the "stopInPinchZonePanel" flag to 1
    stopDistancePanel = distancePanel;  // Assigns the distance value at which the pannel is stoped
  }

  else    // If nothing happens
  {

    OCR2A = 0;                          // Speed 0%
    OCR2B = 0;                          // Opening direction/ doesn't matter which direction
  }

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// OPEN Curtain ///
if (digitalRead(curtainVSwindow) == HIGH)   // If switch is in curtain position(HIGH)
{
// Enter the open statement if the open buttons is presses and the curtain is not in its fully open position
if (digitalRead(openButtonCurtain) == HIGH && fullyOpenCurtain == 0)
{
  fullyClosedCurtain = 0;           // Resets the "fullyClosedCurtain" flag to 0
  OCR1A = 0;                        // Opening Direction
  OCR1B = 255;                      // Speed 100% PWM
  DistanceOpenFunkCurtain();        // Calls the Distance open function
}                                   // Ends the opening statement

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CLOSE Curtain

// Enter the close statement if the close buttons is presses, the curtain is not in its fully closed position,
// if there is no stop in the pinch zone, and if no pinch is detected

else if (digitalRead(closeButtonCurtain) == HIGH && stopInPinchZoneCurtain == 0 && pinchDetectorCurtain == 0 && fullyClosedCurtain ==
 0)
{
  fullyOpenCurtain = 0;             // Resets the "fullyOpenedCurtain" flag to 0
  OCR1A = 255;                      // Closing Direction
  OCR1B = 255;                      // Speed 100% PWM

  if (pinchDetectorCurtain == 0)    // enters if no pinch is detected
  {
    DistanceCloseFunkCurtain();     // Calls the Distance close function
  }

  if (distanceCurtain > UL1Curtain && distanceCurtain <= UL1Curtain + 4)    // Adaptive threshold curve
  {

    if (Time_Per_Revolution_Curtain > Tresh_base_Curtain)  // If The time per revolution is higher than the treshold base will a new 
treshold value be given
    {
      Tresh_base_Curtain = Time_Per_Revolution_Curtain;    // A new base value is assigned to the treshholdcurve that is unique for t
his operation
    }
    // Asigned the Adaptive treshold base value to the C values for the curve
    C1Curtain = Tresh_base_Curtain + C1Curtain_tresh;     // Zone 1
    C2Curtain = Tresh_base_Curtain + C2Curtain_tresh;     // Zone 2
    C3Curtain = Tresh_base_Curtain + C3Curtain_tresh;     // Zone 3
    C4Curtain = Tresh_base_Curtain + C4Curtain_tresh;     // Zone 4
  }

  else if (distanceCurtain >= LL1Curtain && distanceCurtain < UL1Curtain)     // Enters Pinch Zone 1
  {
    // Checks if the treshold curve for Zone 1 is lower than the "time per revolution"
    if (C1Curtain + SensitivityCurtain < Time_Per_Revolution_Curtain)
    {
      pinchDetectorCurtain = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

      // Printind details about the pinch situation
      Serial.print(distanceCurtain);
      Serial.print("; ");
      Serial.print(Time_Per_Revolution_Curtain);
      Serial.print("; ");
      Serial.print("Zone1");
      Serial.print("; ");
      Serial.println(C1Curtain);
    }
  }
  else if (distanceCurtain >= LL2Curtain && distanceCurtain < UL2Curtain)     // Enters Pinch Zone 2
  {
    // Checks if the treshold curve for Zone 2 is lower than the "time per revolution"
    if (C2Curtain + SensitivityCurtain < Time_Per_Revolution_Curtain)
    {
      pinchDetectorCurtain = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

      // Printind details about the pinch situation
      Serial.print(distanceCurtain);
      Serial.print("; ");
      Serial.print(Time_Per_Revolution_Curtain);
      Serial.print("; ");
      Serial.print("Zone2 ");
      Serial.print("; ");
      Serial.println(C2Curtain);
    }
  }
  else if (distanceCurtain >= LL3Curtain && distanceCurtain < UL3Curtain)     // Enters Pinch Zone 3
  {
    // Checks if the treshold curve for Zone 3 is lower than the "time per revolution"
    if (C3Curtain + SensitivityCurtain < Time_Per_Revolution_Curtain)
    {
      pinchDetectorCurtain = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

      // Printind details about the pinch situation
      Serial.print(distanceCurtain);
      Serial.print("; ");
      Serial.print(Time_Per_Revolution_Curtain);
      Serial.print("; ");
      Serial.print("Zone3 ");
      Serial.print("; ");
      Serial.println(C3Curtain);
    }
  }
  else if (distanceCurtain >= LL4Curtain && distanceCurtain < UL4Curtain)     // Enters Pinch Zone 4
  {
    // Checks if the treshold curve for Zone 4 is lower than the "time per revolution"
    if (C4Curtain + SensitivityCurtain < Time_Per_Revolution_Curtain)
    {
      pinchDetectorCurtain = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

      // Printind details about the pinch situation
      Serial.print(distanceCurtain);
      Serial.print("; ");
      Serial.print(Time_Per_Revolution_Curtain);
      Serial.print("; ");
      Serial.print("Zone4 ");
      Serial.print("; ");
      Serial.println(C4Curtain);
    }
  }

} // Stops closing of curtain
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
else if (pinchDetectorCurtain == 1)// If a pinch is detected in the curtain will it change direction and open fully
{

  fullyOpenCurtain = 0;                    // Resets the "fullyOpenCurtain" flag to 0
  OCR1A = 0;                               // Opening Direction
  OCR1B = 255;                             // Speed 100% PWM
  DistanceOpenFunkCurtain();               // Cals the Distance opening function

}                                          // Stops Pinch in curtain statement
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///// Avoids falls pinches by overwriting the pinch stement if the user stops the curtain within the pinch zone
// Enters if Close button is high, stop in pinch zone is high and the curtain is not fully closed

else if (digitalRead(closeButtonCurtain) == HIGH && stopInPinchZoneCurtain == 1 && fullyClosedCurtain == 0)
{
  fullyOpenCurtain = 0;                   // Resets the "fullyOpenCurtain" flag to 0
  OCR1A = 255;                            // Closing Direction
  OCR1B = 255;                            // Speed 100% PWM
  DistanceCloseFunkCurtain();             // Cals the Distance close function
}                                         // Stops Stop in pinch zone statement
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////// Allow the user to stop the curtain within the pinch zone
// Enters if both buttons are low and the curtain is in the pinch zone

else if (digitalRead(openButtonCurtain) == LOW && digitalRead(closeButtonCurtain) == LOW && distanceCurtain > LL4Curtain && distanceC
urtain < UL1Curtain)
{
  OCR1A = 255;                            // Closing direction/ doesn't matter which direction
  OCR1B = 0;                              // Speed 0% PWM
  stopInPinchZoneCurtain = 1;             // Sets the "stopInPinchZoneCurtain" flag to 1
  stopDistanceCurtain = distanceCurtain;  // Assigns the distance value at which the curtain is stoped
}                                         // Stops Stop in pinch zone statement



else                                      // If nothing happens in Curtain
{
  OCR1A = 255;                            // Closing direction/ doesn't matter which direction
  OCR1B = 0;                              // Speed 0% PWM
}                                         // Stops if nothing happens statement

}                                         // Stops switch statement
//////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////// OPEN Window /////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////
else if (digitalRead(curtainVSwindow) == LOW)   // If switch is in Window position(LOW)
{
// Enter the open statement if the open buttons is presses and the window is not in its fully open position
  if (digitalRead(openButtonWindow) == HIGH && fullyOpenWindow == 0)
  {
      fullyClosedWindow = 0;                // Resets the "fullyClosedWindow" flag to 0
      OCR1A = 0;                            // Opening Direction
      OCR1B = 255;                          // Speed 100% PWM
      DistanceOpenFunkWindow();             // Calls the Distance open function
  }

//////////////////////////////////////////////////////////////////////////////////////////////
// CLOSE Window

// Enter the close statement if the close buttons is presses, the window is not in its fully closed position,
// if there is no stop in the pinch zone, and if no pinch is detected

  else if (digitalRead(closeButtonWindow) == HIGH && stopInPinchZoneWindow == 0 && pinchDetectorWindow == 0 && fullyClosedWindow == 0
)
  {
    fullyOpenWindow = 0;                    // Resets the "fullyOpenWindow" flag to 0
    OCR1A = 255;                            // Closing Direction
    OCR1B = 255;                            // Speed 100% PWM


    if (pinchDetectorWindow == 0)           // enters if no pinch is detected
    {
      DistanceCloseFunkWindow();            // Calls the Distance close function
    }

    if (distanceWindow > UL1Window && distanceWindow <= UL1Window + 4)     // Adaptive threshold curve
    {

      if (Time_Per_Revolution_Window > Tresh_base_Window)  // If The time per revolution is higher than the treshold base will a new
treshold value be given
      {

        Tresh_base_Window = Time_Per_Revolution_Window;    // A new base value is assigned to the treshholdcurve that is unique for t
his operation
      }
      // Asigned the Adaptive treshold base value to the C values for the curve

      C1Window = Tresh_base_Window + C1Window_thresh + sensitivityWindow;     // Zone 1
      C2Window = Tresh_base_Window + C2Window_thresh + sensitivityWindow;     // Zone 2
      C3Window = Tresh_base_Window + C3Window_thresh + sensitivityWindow;     // Zone 3
      C4Window = Tresh_base_Window + C4Window_thresh + sensitivityWindow;     // Zone 4
      C5Window = Tresh_base_Window + C5Window_thresh + sensitivityWindow;     // Zone 5
      C6Window = Tresh_base_Window + C6Window_thresh + sensitivityWindow;     // Zone 6
    }
    else if (distanceWindow >= LL1Window && distanceWindow < UL1Window)     // Enters Pinch Zone 1
    {
      // Checks if the treshold curve for Zone 1 is lower than the "time per revolution"
      if (A1Window * (UL1Window - distanceWindow) * (UL1Window - distanceWindow) + B1Window * (UL1Window - distanceWindow) + C1Window
 - pinchSensitivityWindow < Time_Per_Revolution_Window)
      {
       pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

//         Printind details about the pinch situation
        Serial.print(round(A1Window * (UL1Window - distanceWindow) * (UL1Window - distanceWindow) + B1Window * (UL1Window - distanceW
indow) + C1Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.print("Zone1");

      }
    }
    else if (distanceWindow >= LL2Window && distanceWindow < UL2Window)     // Enters Pinch Zone 2
    {
      // Checks if the treshold curve for Zone 2 is lower than the "time per revolution"
      if (A2Window * (UL2Window - distanceWindow) * (UL2Window - distanceWindow) + B2Window * (UL2Window - distanceWindow) + C2Window
 - pinchSensitivityWindow < Time_Per_Revolution_Window)
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A2Window * (UL2Window - distanceWindow) * (UL2Window - distanceWindow) + B2Window * (UL2Window - distanceW
indow) + C2Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone2");

      }
    }
    else if (distanceWindow >= LL3Window && distanceWindow < UL3Window)     // Enters Pinch Zone 3
    {
      // Checks if the treshold curve for Zone 3 is lower than the "time per revolution"
      if (A3Window * (UL3Window - distanceWindow) * (UL3Window - distanceWindow) + B3Window * (UL3Window - distanceWindow) + C3Window
 - pinchSensitivityWindow < Time_Per_Revolution_Window)
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A3Window * (UL3Window - distanceWindow) * (UL3Window - distanceWindow) + B3Window * (UL3Window - distanceW
indow) + C3Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone3");

      }
    }
    else if (distanceWindow >= LL4Window && distanceWindow < UL4Window)     // Enters Pinch Zone 4
    {
      // Checks if the treshold curve for Zone 4 is lower than the "time per revolution"
      if ( A4Window * (UL4Window - distanceWindow) * (UL4Window - distanceWindow) + B4Window * (UL4Window - distanceWindow) + C4Windo
w - pinchSensitivityWindow < Time_Per_Revolution_Window)
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A4Window * (UL4Window - distanceWindow) * (UL4Window - distanceWindow) + B4Window * (UL4Window - distanceW
indow) + C4Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone4");
      }
    }
    else if (distanceWindow >= LL5Window && distanceWindow < UL5Window)     // Enters Pinch Zone 5
    {
      // Checks if the treshold curve for Zone 5 is lower than the "time per revolution"
      if ( A5Window * (UL5Window - distanceWindow) * (UL5Window - distanceWindow) + B5Window * (UL5Window - distanceWindow) + C5Windo
w - pinchSensitivityWindow < Time_Per_Revolution_Window)
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A5Window * (UL5Window - distanceWindow) * (UL5Window - distanceWindow) + B5Window * (UL5Window - distanceW
indow) + C5Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone5");

      }
    }
    else if (distanceWindow >= LL6Window && distanceWindow < UL6Window)     // Enters Pinch Zone 6
    {
      // Checks if the treshold curve for Zone 6 is lower than the "time per revolution"
      if (A6Window * (UL6Window - distanceWindow) * (UL6Window - distanceWindow) + B6Window * (UL6Window - distanceWindow) + C6Window
 - pinchSensitivityWindow < Time_Per_Revolution_Window)
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1
```

```
        // Printind details about the pinch situation
        Serial.print(round(A6Window * (UL6Window - distanceWindow) * (UL6Window - distanceWindow) + B6Window * (UL6Window - distanceW
indow) + C6Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone6");
        Serial.print("; ");
        Serial.println(C6Window);
      }
    }
  }                                  // Stops Close statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
  else if (pinchDetectorWindow == 1)// If a pinch is detected in the window will it change direction and open fully
  {

    fullyOpenWindow = 0;             // Resets the "fullyOpenWindow" flag to 0
    OCR1A = 0;                       // Opening Direction
    OCR1B = 255;                     // Speed 100% PWM
    DistanceOpenFunkWindow();        // Cals the Distance opening function
  }                                  // Stops pinch in Window statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///// Avoids falls pinches by overwriting the pinch stement if the user stops the window within the pinch zone
// Enters if Close button is high, stop in pinch zone is high and the window is not fully closed

  else if (digitalRead(closeButtonWindow) == HIGH && stopInPinchZoneWindow == 1 && fullyClosedWindow == 0)
  {
    fullyOpenWindow = 0;             // Resets the "fullyOpenWindow" flag to 0
    OCR1A = 255;                     // Closing Direction
    OCR1B = 255;                     // Speed 100% PWM
    DistanceCloseFunkWindow();       // Cals the Distance close function
  }                                  // Stops Stop in pinch zone statement

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////// Allow the user to stop the window within the pinch zone
// Enters if both buttons are low and the window is in the pinch zone

  else if (digitalRead(openButtonWindow) == LOW && digitalRead(closeButtonWindow) == LOW  && distanceWindow > LL6Window && distanceWi
ndow < UL1Window)
  {

    OCR1A = 255;                            // Closing direction/ doesn't matter which direction
    OCR1B = 0;                              // Speed 0% PWM
    stopInPinchZoneWindow = 1;              // Sets the "stopInPinchZoneWindow" flag to 1
    stopDistanceWindow = distanceWindow;    // Assigns the distance value at which the window is stoped
  }                                         // Stops Stop in pinch zone statement

  else                                      // If nothing happens in Window
  {
    OCR1A = 255;                            // Closing direction/ doesn't matter which direction
    OCR1B = 0;                              // Speed 0% PWM
  }                                         // Stops if nothing happens statement

}                                           // Stops switch statement

} //ENDS  VOID LOOP

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculate the pacing in Panel
void PaceFunkPanel()
{
  Time_Between_Hs_Input_Panel = hs1TimePanel - hs2TimePanel;           // Inverse of RPM
  Time_Between_Hs_Input_Panel = abs(Time_Between_Hs_Input_Panel);      // Take the absolute value

  Time_Per_Revolution_Panel   = Time_Between_Hs_Input_Panel - Time_Between_Hs_Input_Panel_Old;   // Checks the difference between the
 old and new time difference between hs input
  Time_Per_Revolution_Panel   = abs(Time_Per_Revolution_Panel);        // Take the absolute value

  hsDetectPanel = 0;                                                   // Resets Hs detect to 0

  Time_Between_Hs_Input_Panel_Old  = Time_Between_Hs_Input_Panel;      // Updated the Time_Between_Hs_Input_Panel_Old

// Printing of the treshold curve
   if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL1Panel && distancePanel < UL1Panel)
    {
          Serial.print(round(A1Panel * (UL1Panel - distancePanel) * (UL1Panel - distancePanel) + B1Panel * (UL1Panel - distancePanel)
 + C1Panel - pinchSensitivityPanel));
          Serial.print("; ");
    }

    else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL2Panel && distancePanel < UL2Panel)
    {
          Serial.print(round(A2Panel * (UL2Panel - distancePanel) * (UL2Panel - distancePanel) + B2Panel * (UL2Panel - distancePanel)
 + C2Panel - pinchSensitivityPanel));
          Serial.print("; ");
    }

    else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL3Panel && distancePanel < UL3Panel)
    {
          Serial.print(round(A3Panel * (UL3Panel - distancePanel) * (UL3Panel - distancePanel) + B3Panel * (UL3Panel - distancePanel)
 + C3Panel - pinchSensitivityPanel));
          Serial.print("; ");
    }
    else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL4Panel && distancePanel < UL4Panel)
    {
          Serial.print(round(A4Panel * (UL4Panel - distancePanel) * (UL4Panel - distancePanel) + B4Panel * (UL4Panel - distancePanel)
 + C4Panel - pinchSensitivityPanel));
          Serial.print("; ");
    }

    else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL5Panel && distancePanel < UL5Panel)
    {
         Serial.print(round(A5Panel * (UL5Panel - distancePanel) * (UL5Panel - distancePanel) + B5Panel * (UL5Panel - distancePanel)
+ C5Panel - pinchSensitivityPanel));
         Serial.print("; ");
    }

    else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL6Panel && distancePanel < UL6Panel)
    {
         Serial.print(round(A6Panel * (UL6Panel - distancePanel) * (UL6Panel - distancePanel) + B6Panel * (UL6Panel - distancePanel)
+ C6Panel - pinchSensitivityPanel));
         Serial.print("; ");
    }

    else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL7Panel && distancePanel < UL7Panel)
    {
        Serial.print(round(A7Panel * (UL7Panel - distancePanel) * (UL7Panel - distancePanel) + B7Panel * (UL7Panel - distancePanel) +
 C7Panel - pinchSensitivityPanel));
        Serial.print("; ");
    }

    else
    {
        Serial.print(0);
        Serial.print("; ");
    }
        // Printing distance, Time_Per_Revolution_Panel and the treshbase
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the opening of the panel

void DistanceOpenFunkPanel()
{
  progTimePanel = millis();                           // Updated the program time

  if (oneTimeCheckPanel == 0)                         // Enters When there is a hall sensor input
  {
    fullyOpenedTimerPanel = progTimePanel;            // When there is a hall sensor input is the timer reseted
    oneTimeCheckPanel = 1;                            // Sets flag to 1
  }

  // To detect if the panel is at its fully open position
  // Enters if the difference between the program time & the fully open time is less than the specified hard stop time
  if (progTimePanel - fullyOpenedTimerPanel <= hardStopTimePanel)
  {
    if (hsDetectPanel >= hSensitivityPanel)           // Enters when a hall sensor is detected
    {
      distancePanel = distancePanel + confirmerPanel; // Increases the distance
      confirmerPanel = 0;                             // Resets the confirmer
      PaceFunkPanel();                                // Calls the pace function to calculate the time per revolution
      bufferClosePanel = 0;                           // Resets the closing buffer for the hard stop
      fullyClosedPanel = 0;                           // Tells the system that the panel has not reached it fully closed position
      oneTimeCheckPanel = 0;                          // Resets one time check
      bufferOpenPanel = 0;                            // Resets the opening buffer for the hard stop
    }
  }

  // Enters if the difference between the program time & the fully open time is more than the specified hard stop time (to longe betw
een hall sensor input)
  else if (progTimePanel - fullyOpenedTimerPanel > hardStopTimePanel)
  {
    if (bufferOpenPanel == 0) // Enters if the buffer is 0 --> first time it is to long between Hs inputs
    {
      Serial.println(" --- buffer --- ");             // Prints buffer
      distancePanel = distancePanel + confirmerPanel; // Ingreases the distance
      confirmerPanel = 0;                             // Resets the confirmer
      PaceFunkPanel();                                // Calls the pace function to calculate the time per revolution
      fullyClosedPanel = 0;                           // Tells the system that the panel has not reached it fully closed position
      oneTimeCheckPanel = 0;                          // Resets one time check
      bufferOpenPanel = 1;                            // Sets buffer to 1 so that it doesnt enter tha same statement next time
    }
    else
    {
      Serial.println(" --- PANEL FULLY OPEN --- ");   // Prints That the panel is fully open
      pinchDetectorPanel = 0;                         // Resets the pinch detector to 0 so that the panel can be operated as normal a
gain if a pinch was detected
      distancePanel = fullyOpenDistancePanel;         // Sets the distance to the fully open distance
      progTimePanel = 0;                              // Resets the program time to 0
      fullyOpenedTimerPanel = 0;                      // Resets the fully open timer
      oneTimeCheckPanel = 0;                          // Resets one time check
      fullyOpenPanel = 1;                             // Tells the system that the panel is fully open
      fullyClosedPanel = 0;                           // Tells the system that the panel is NOT fully closed
    }
  }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the closing of the panel
void DistanceCloseFunkPanel()
{
  progTimePanel = millis();                           // Updated the program time
  if (oneTimeCheckPanel == 0)                         // Enters When there is a hall sensor input
  {
    fullyClosedTimerPanel = progTimePanel;            // When there is a hall sensor input is the timer reseted
    oneTimeCheckPanel = 1;                            // Sets flag to 1
  }

  // To detect if the panel is at its fully closed position
  // Enters if the difference between the program time & the fully close time is less than the specified hard stop time
  if (progTimePanel - fullyClosedTimerPanel <= hardStopTimePanel)
  {
    if (hsDetectPanel >= hSensitivityPanel)           // Enters when a hall sensor is detected
    {
      distancePanel = distancePanel - confirmerPanel; // Decreases the distance
      confirmerPanel = 0;                             // Resets the confirmer
      PaceFunkPanel();                                // Calls the pace function to calculate the time per revolution
      bufferOpenPanel = 0;                            // Resets the Opening buffer for the hard stop
      fullyOpenPanel = 0;                             // Tells the system that its NOT fully open
      oneTimeCheckPanel = 0;                          // Resets one time check
      bufferClosePanel = 0;                           // Resets the buffer for the closing Hard Stop

      if (distancePanel < stopDistancePanel - 20)     // Enter is if the panel has moved more than 20 steps since the stop in pinch z
one
      {
        stopInPinchZonePanel = 0;                     // Resets stopInPinchZonePanel flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
  }
  // Enters if the difference between the program time & the fully closed time is more than the specified hard stop time (to longe be
tween hall sensor input)
  else if (progTimePanel - fullyClosedTimerPanel > hardStopTimePanel)
  {
    if (bufferClosePanel == 0)                        // Enters if the buffer is 0 --> first time it is to long between Hs inputs
    {
      Serial.println(" --- buffer --- ");             // Prints buffer
      distancePanel = distancePanel - confirmerPanel; // Decreases the distance
      confirmerPanel = 0;                             // Resets the confirmer
      PaceFunkPanel();                                // Calls the pace function to calculate the time per revolution
      fullyOpenPanel = 0;                             // Tells the system that the panel has not reached it fully open position
      oneTimeCheckPanel = 0;                          // Resets one time check
      bufferClosePanel = 1;                           // Sets buffer to 1 so that it doesnt enter tha same statement next time
      if (distancePanel < stopDistancePanel - 20)     // Enter is if the panel has moved more than 20 steps since the stop in pinch z
one
      {
        stopInPinchZonePanel = 0;                     // Resets stopInPinchZonePanel flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
    else
    {
      Serial.println(" --- PANEL FULLY CLOSED --- "); // Prints That the panel is fully closed
      distancePanel = fullyClosedDistancePanel;       // Sets the distance to the fully closed distance
      progTimePanel = 0;                              // Resets the program time to 0
      fullyClosedTimerPanel = 0;                      // Resets the fully closed timer
      oneTimeCheckPanel = 0;                          // Resets one time check
      fullyClosedPanel = 1;                           // Tells the system that the panel is fully closed
      fullyOpenPanel = 0;                             // Tells the system that the panel is NOT fully open
    }
  }
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funktion that Identifies input from Hall sensor 1 in Panel
void magnet_detect1Panel()            // The funtions is called by the interrupt when a signal is recived from hall sensor 1
{
  // Enter if any of the buttons are pressed, if there has been a pinch and if the panel is not in its fully closed/open position
  if ((digitalRead(openButtonPanel) == HIGH || digitalRead(closeButtonPanel) == HIGH || pinchDetectorPanel == 1) && fullyOpenPanel ==
 0 && fullyClosedPanel == 0)
  {
    hsDetectPanel++;                  // Steps upp hsDetect
    confirmerPanel++;                 // Steps upp the confirmer
    hs1TimePanel = micros();          // Updates the hall sensor 1 timer
  }
}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funktion that Identifies input from Hall sensor 2 in Panel
void magnet_detect2Panel()            // The funtions is called by the interrupt when a signal is recived from hall sensor 2
{
 // Enter if any of the buttons are pressed, if there has been a pinch and if the panel is not in its fully closed/open position
  if ((digitalRead(openButtonPanel) == HIGH || digitalRead(closeButtonPanel) == HIGH || pinchDetectorPanel == 1)&& fullyOpenPanel ==
0 && fullyClosedPanel == 0)
  {
    hsDetectPanel++;                  // Steps upp hsDetect
    confirmerPanel++;                 // Steps upp the confirmer
    hs2TimePanel = micros();          // Updates the hall sensor 2 timer
  }
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculate the pacing in the Curtain
void paceCurtainFunk()
{
  Time_Between_Hs_Input_Curtain= hs1CurtainTime - hs2CurtainTime;           // Inverse of RPM
  Time_Between_Hs_Input_Curtain= abs(Time_Between_Hs_Input_Curtain);        // Take the absolute value

  Time_Per_Revolution_Curtain = Time_Between_Hs_Input_Curtain- Time_Between_Hs_Input_CurtainOld;   // Checks the difference between t
he old and new time difference between hs input
  Time_Per_Revolution_Curtain = abs(Time_Per_Revolution_Curtain);           // Take the absolute value

  hsDetectCurtain = 0;                                                      // Resets Hs detect to 0

  Time_Between_Hs_Input_CurtainOld = Time_Between_Hs_Input_Curtain;         // Updated the Time_Between_Hs_Input_CurtainOld

  // Printing distance and Time_Per_Revolution_Curtain
  Serial.print(distanceCurtain);
  Serial.print("; ");
  Serial.println(Time_Per_Revolution_Curtain);
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the opening of the curtain
void DistanceOpenFunkCurtain()
{
  progTimeCurtain = millis();                                   // Updated the program time
  if (oneTimeCheckCurtain == 0)                                 // Enters When there is a hall sensor input
  {
    fullyOpenedTimerCurtain = progTimeCurtain;                  // When there is a hall sensor input is the timer reseted
    oneTimeCheckCurtain = 1;                                    // Sets flag to 1
  }

  // To detect if the curtain is at its fully open position
  // Enters if the difference between the program time & the fully open time is less than the specified hard stop time
  if (progTimeCurtain - fullyOpenedTimerCurtain <= hardStopTimeCurtain)
  {
    if (hsDetectCurtain >= hSensitivityCurtain)                 // Enters when a hall sensor is detected
    {
      distanceCurtain = distanceCurtain + confirmerCurtain;     // Increases the distance
      confirmerCurtain = 0;                                     // Resets the confirmer
      paceCurtainFunk();                                        // Calls the pace function to calculate the time per revolution
      bufferCloseCurtain = 0;                                   // Resets the closing buffer for the hard stop
      fullyClosedCurtain = 0;                                   // Tells the system that the curtain has not reached it fully closed
position
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      bufferOpenCurtain = 0;                                    // Resets the opening buffer for the hard stop
    }
  }

 // Enters if the difference between the program time & the fully open time is more than the specified hard stop time (to longe betwe
en hall sensor input)
 else if (progTimeCurtain - fullyOpenedTimerCurtain > hardStopTimeCurtain)
  {
    if (bufferOpenCurtain == 0)                                 // Enters if the buffer is 0 --
> first time it is to long between Hs inputs
    {
      Serial.println(" ---buffer--- ");                         // Prints buffer
      distanceCurtain = distanceCurtain + confirmerCurtain;     // Ingreases the distance
      confirmerCurtain = 0;                                     // Resets the confirmer
      paceCurtainFunk();                                        // Calls the pace function to calculate the time per revolution
      fullyClosedCurtain = 0;                                   // Tells the system that the curtain has not reached it fully closed
position
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      bufferOpenCurtain = 1;                                    // Sets buffer to 1 so that it doesnt enter tha same statement next t
ime
    }
    else
    {
      Serial.println(" --- CURTAIN FULLY OPEN --- ");           // Prints That the curtain is fully open
      pinchDetectorCurtain = 0;                                 // Resets the pinch detector to 0 so that the curtain can be operated
 as normal again if a pinch was detected
      distanceCurtain = fullyOpenDistanceCurtain;               // Sets the distance to the fully open distance
      progTimeCurtain = 0;                                      // Resets the program time to 0
      fullyOpenedTimerCurtain = 0;                              // Resets the fully open timer
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      fullyOpenCurtain = 1;                                     // Tells the system that the curtain is fully open
      fullyClosedCurtain = 0;                                   // Tells the system that the curtain is NOT fully closed
    }
  }
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the closing of the curtain
void DistanceCloseFunkCurtain()
{
  progTimeCurtain = millis();                                   // Updated the program time
  if (oneTimeCheckCurtain == 0)                                 // Enters When there is a hall sensor input
  {
    fullyClosedTimerCurtain = progTimeCurtain;                  // When there is a hall sensor input is the timer reseted
    oneTimeCheckCurtain = 1;                                    // Sets flag to 1
  }

  // To detect if the curtain is at its fully closed position
  // Enters if the difference between the program time & the fully close time is less than the specified hard stop time
  if (progTimeCurtain - fullyClosedTimerCurtain <= hardStopTimeCurtain)
  {
    if (hsDetectCurtain >= hSensitivityCurtain)                 // Enters when a hall sensor is detected
    {
      distanceCurtain = distanceCurtain - confirmerCurtain;     // Decreases the distance
      confirmerCurtain = 0;                                     // Resets the confirmer
      paceCurtainFunk();                                        // Calls the pace function to calculate the time per revolution
      bufferOpenCurtain = 0;                                    // Resets the Opening buffer for the hard stop
      fullyOpenCurtain = 0;                                     // Tells the system that the curtain is NOT fully open
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      bufferCloseCurtain = 0;                                   // Resets the buffer for the closing Hard Stop

      if (distanceCurtain < stopDistanceCurtain - 20)           // Enter is if the curtain has moved more than 20 steps since the sto
p in pinch zone
      {
        stopInPinchZoneCurtain = 0;                             // Resets stopInPinchZoneCurtain flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
  }
 // Enters if the difference between the program time & the fully closed time is more than the specified hard stop time (to longe bet
ween hall sensor input)
 else if (progTimeCurtain - fullyClosedTimerCurtain > hardStopTimeCurtain)
  {
    if (bufferCloseCurtain == 0)                                // Enters if the buffer is 0 --
> first time it is to long between Hs inputs
    {
      Serial.println(" --- buffer --- ");                       // Prints buffer
      distanceCurtain = distanceCurtain - confirmerCurtain;     // Decreases the distance
      confirmerCurtain = 0;                                     // Resets the confirmer
      paceCurtainFunk();                                        // Calls the pace function to calculate the time per revolution
      fullyOpenCurtain = 0;                                     // Tells the system that the curtain has not reached it fully open po
sition
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      bufferCloseCurtain = 1;                                   // Sets buffer to 1 so that it doesnt enter tha same statement next t
ime

      if (distanceCurtain < stopDistanceCurtain - 20)           // Enter is if the curtain has moved more than 20 steps since the sto
p in pinch zone
      {
        stopInPinchZoneCurtain = 0;                             // Resets stopInPinchZoneCurtain flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
    else
    {
      Serial.println(" --- CURTAIN FULLY CLOSED --- ");         // Prints That the curtain is fully closed
      distanceCurtain = fullyClosedDistanceCurtain;             // Sets the distance to the fully closed distance
      progTimeCurtain = 0;                                      // Resets the program time to 0
      fullyClosedTimerCurtain = 0;                              // Resets the fully closed timer
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      fullyClosedCurtain = 1;                                   // Tells the system that the curtain is fully closed
      fullyOpenCurtain = 0;                                     // Tells the system that the curtain is NOT fully open
    }
  }
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funktion that Identifies input from Hall sensor 1 in Curtain
void magnet_detect1Curtain()            // The funtions is called by the interrupt when a signal is recived from hall sensor 1
{

  // Enter if any of the buttons are pressed, if there has been a pinch and if the curtain is not in its fully closed/open position
  if ((digitalRead(openButtonCurtain) == HIGH || digitalRead(closeButtonCurtain) == HIGH || pinchDetectorCurtain == 1) && fullyOpenCu
rtain == 0 && fullyClosedCurtain == 0)
  {
    hsDetectCurtain++;                  // Steps upp hsDetect
    confirmerCurtain++;                 // Steps upp the confirmer
    hs1CurtainTime = micros();          // Updates the hall sensor 1 timer
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funktion that Identifies input from Hall sensor 2 in Curtain
void magnet_detect2Curtain()            // The funtions is called by the interrupt when a signal is recived from hall sensor 2
{

 // Enter if any of the buttons are pressed, if there has been a pinch and if the curtain is not in its fully closed/open position
  if ((digitalRead(openButtonCurtain) == HIGH || digitalRead(closeButtonCurtain) == HIGH || pinchDetectorCurtain == 1) && fullyOpenCu
rtain == 0 && fullyClosedCurtain == 0)  {

    hsDetectCurtain++;                  // Steps upp hsDetect
    confirmerCurtain++;                 // Steps upp the confirmer
    hs2CurtainTime = micros();          // Updates the hall sensor 2 timer
  }
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculate the pacing in the front Window
void PaceFunkWindow()
{
  Time_Between_Hs_Input_Window = hs1TimeWindow - hs2TimeWindow;             // Inverse of RPM
  Time_Between_Hs_Input_Window = abs(Time_Between_Hs_Input_Window);         // Take the absolute value

  Time_Per_Revolution_Window = Time_Between_Hs_Input_Window - Time_Between_Hs_Input_WindowOld;// Checks the difference between the ol
d and new time difference between hs input
  Time_Per_Revolution_Window = abs(Time_Per_Revolution_Window);             // Take the absolute value

  hsDetectWindow = 0;                                                       // Resets Hs detect to 0

  Time_Between_Hs_Input_WindowOld = Time_Between_Hs_Input_Window;           // Updated the Time_Between_Hs_Input_WindowOld

// Printing of the treshold curve
   if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL1Window && distanceWindow < UL1Window)
    {
          Serial.print(round(A1Window * (UL1Window - distanceWindow) * (UL1Window - distanceWindow) + B1Window * (UL1Window - distanc
eWindow) + C1Window - pinchSensitivityWindow));
```

```
          Serial.print("; ");
    }

    else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL2Window && distanceWindow < UL2Window)
    {
          Serial.print(round(A2Window * (UL2Window - distanceWindow) * (UL2Window - distanceWindow) + B2Window * (UL2Window - distanc
eWindow) + C2Window - pinchSensitivityWindow));
          Serial.print("; ");
    }

    else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL3Window && distanceWindow < UL3Window)
    {
          Serial.print(round(A3Window * (UL3Window - distanceWindow) * (UL3Window - distanceWindow) + B3Window * (UL3Window - distanc
eWindow) + C3Window - pinchSensitivityWindow));
          Serial.print("; ");
    }

    else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL4Window && distanceWindow < UL4Window)
    {
          Serial.print(round(A4Window * (UL4Window - distanceWindow) * (UL4Window - distanceWindow) + B4Window * (UL4Window - distanc
eWindow) + C4Window - pinchSensitivityWindow));
          Serial.print("; ");
    }

    else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL5Window && distanceWindow < UL5Window)
    {
         Serial.print(round(A5Window * (UL5Window - distanceWindow) * (UL5Window - distanceWindow) + B5Window * (UL5Window - distance
Window) + C5Window - pinchSensitivityWindow));
         Serial.print("; ");
    }

    else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL6Window && distanceWindow < UL6Window)
    {
         Serial.print(round(A6Window * (UL6Window - distanceWindow) * (UL6Window - distanceWindow) + B6Window * (UL6Window - distance
Window) + C6Window - pinchSensitivityWindow));
         Serial.print("; ");
    }

    else
    {
        Serial.print(0);
        Serial.print("; ");
    }
        // Printing distance and Time_Per_Revolution_Window
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.println(Time_Per_Revolution_Window);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Calculated the distance during the opening of the Window
void DistanceOpenFunkWindow()
{
  progTimeWindow = millis();                                     // Updated the program time
  if (oneTimeCheckWindow == 0)                                   // Enters When there is a hall sensor input
  {
    fullyOpenedTimerWindow = progTimeWindow;                    // When there is a hall sensor input is the timer reseted
    oneTimeCheckWindow = 1;                                      // Sets flag to 1
  }

  // To detect if the front window is at its fully open position
  // Enters if the difference between the program time & the fully open time is less than the specified hard stop time
   if (progTimeWindow - fullyOpenedTimerWindow <= hardStopTimeWindow)
  {
    if (hsDetectWindow >= hSensitivityWindow)                   // Enters when a hall sensor is detected
    {
      distanceWindow = distanceWindow + confirmerWindow;       // Increases the distance
      confirmerWindow = 0;                                       // Resets the confirmer
      PaceFunkWindow();                                          // Calls the pace function to calculate the time per revolution
      bufferCloseWindow = 0;                                     // Resets the closing buffer for the hard stop
      fullyClosedWindow = 0;                                     // Tells the system that the front window has not reached it fully clo
sed position
      oneTimeCheckWindow = 0;                                    // Resets one time check
      bufferOpenWindow = 0;                                      // Resets the opening buffer for the hard stop
    }
  }

 // Enters if the difference between the program time & the fully open time is more than the specified hard stop time (to longe betwe
en hall sensor input)
  else if (progTimeWindow - fullyOpenedTimerWindow > hardStopTimeWindow)
  {
    if (bufferOpenWindow == 0)                                   // Enters if the buffer is 0 --
> first time it is to long between Hs inputs
    {
//      Serial.println(" ---buffer--- ");                          // Prints buffer
      distanceWindow = distanceWindow + confirmerWindow;         // Ingreases the distance
      confirmerWindow = 0;                                       // Resets the confirmer
      PaceFunkWindow();                                          // Calls the pace function to calculate the time per revolution
      fullyClosedWindow = 0;                                     // Tells the system that the front window has not reached it fully clo
sed position
      oneTimeCheckWindow = 0;                                    // Resets one time check
      bufferOpenWindow = 1;                                      // Sets buffer to 1 so that it doesnt enter tha same statement next ti
me
    }
    else
    {
//      Serial.println(" --- WINDOW FULLY OPEN --- ");             // Prints That the front Window is fully open
```

```
      pinchDetectorWindow = 0;                                   // Resets the pinch detector to 0 so that the window can be operated a
s normal again if a pinch was detected
      distanceWindow = fullyOpenDistanceWindow;                  // Sets the distance to the fully open distance
      progTimeWindow = 0;                                        // Resets the program time to 0
      fullyOpenedTimerWindow = 0;                                // Resets the fully open timer
      oneTimeCheckWindow = 0;                                    // Resets one time check
      fullyClosedWindow = 0;                                     // Tells the system that the window is NOT fully closed
      fullyOpenWindow = 1;                                       // Tells the system that the window is fully open
    }
  }
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the closing of the Window
void DistanceCloseFunkWindow()
{
  progTimeWindow = millis();                                     // Updated the program time
  if (oneTimeCheckWindow == 0)                                   // Enters When there is a hall sensor input
  {
    fullyClosedTimerWindow = progTimeWindow;                    // When there is a hall sensor input is the timer reseted
    oneTimeCheckWindow = 1;                                      // Sets flag to 1
  }

  // To detect if the front window is at its fully closed position
  // Enters if the difference between the program time & the fully close time is less than the specified hard stop time
  if (progTimeWindow - fullyClosedTimerWindow <= hardStopTimeWindow)
  {
    if (hsDetectWindow >= hSensitivityWindow)                    // Enters when a hall sensor is detected
    {
      distanceWindow = distanceWindow - confirmerWindow;         // Decreases the distance
      confirmerWindow = 0;                                       // Resets the confirmer
      PaceFunkWindow();                                          // Calls the pace function to calculate the time per revolution
      bufferOpenWindow = 0;                                      // Resets the Opening buffer for the hard stop
      fullyOpenWindow = 0;                                       // Tells the system that the window is NOT fully open
      oneTimeCheckWindow = 0;                                    // Resets one time check
      bufferCloseWindow = 0;                                     // Resets the buffer for the closing Hard Stop

      if (distanceWindow < stopDistanceWindow - 20)              // Enter is if the window has moved more than 20 steps since the stop
in pinch zone
      {
        stopInPinchZoneWindow = 0;                               // Resets stopInPinchZoneWindow flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
  }
// Enters if the difference between the program time & the fully closed time is more than the specified hard stop time (to longe betw
een hall sensor input)
else if (progTimeWindow - fullyClosedTimerWindow > hardStopTimeWindow)
  {
    if (bufferCloseWindow == 0)                                  // Enters if the buffer is 0 --
> first time it is to long between Hs inputs
    {
//      Serial.println(" --- buffer --- ");                         // Prints buffer
      distanceWindow = distanceWindow - confirmerWindow;       // Decreases the distance
      confirmerWindow = 0;                                       // Resets the confirmer
      PaceFunkWindow();                                          // Calls the pace function to calculate the time per revolution
      fullyOpenWindow = 0;                                       // Tells the system that the window has not reached it fully open posi
tion
      oneTimeCheckWindow = 0;                                    // Resets one time check
      bufferCloseWindow = 1;                                     // Sets buffer to 1 so that it doesnt enter tha same statement next ti
me

      if (distanceWindow < stopDistanceWindow - 20)             // Enter is if the window has moved more than 20 steps since the stop i
n pinch zone
      {
        stopInPinchZoneWindow = 0;                               // Resets stopInPinchZoneWindow flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
    else
    {
//      Serial.println(" --- WINDOW FULLY CLOSED --- ");          // Prints That the window is fully closed
      distanceWindow = fullyClosedDistanceWindow;               // Sets the distance to the fully closed distance
      progTimeWindow = 0;                                        // Resets the program time to 0
      fullyClosedTimerWindow = 0;                                // Resets the fully closed timer
      oneTimeCheckWindow = 0;                                    // Resets one time check
      fullyOpenWindow = 0;                                       // Tells the system that the window is NOT fully open
      fullyClosedWindow = 1;                                     // Tells the system that the window is fully closed
    }
  }
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funktion that Identifies input from Hall sensor 1 in the front window
void magnet_detect1Window()            // The funtions is called by the interrupt when a signal is recived from hall sensor 1
{
   // Enter if any of the buttons are pressed, if there has been a pinch and if the window is not in its fully closed/open position
  if (digitalRead(openButtonWindow) == HIGH || digitalRead(closeButtonWindow) == HIGH || pinchDetectorWindow == 1)
  {
    hsDetectWindow++;                  // Steps upp hsDetect
    confirmerWindow++;                 // Steps upp the confirmer
    hs1TimeWindow = micros();          // Updates the hall sensor 1 timer
  }
}
```

Appendix G

Arduino code concept 2.

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This program uses the hall sensor input from the panel, curtain and front window motors to determine their position and speed //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////// Initiate Parameters Panel //////////////////////////////////////////////////////
volatile byte hsDetectPanel = 0;         // Hall Sensor detection in the panel
volatile byte confirmerPanel = 0;        // Vairable used to help determine the distance
const byte hSensitivityPanel = 1;        // How sensitive will the system be / the readings of the hall sensors

unsigned long hs1TimePanel = 0;          // Time when hall sensor 1 is detected
unsigned long hs2TimePanel = 0;          // Time when hall sensor 2 is detected

// Parameters for determining if the panel has reached its end position
long progTimePanel = 0;                  // A timer to start timing when the user presses the button
long fullyOpenedTimerPanel = 0;          // A timer to measure time between the last hall sensor input and program time while closing
 the panel
long fullyClosedTimerPanel = 0;          // A timer to measure time between the last hall sensor input and program time while opening
 the panel
byte oneTimeCheckPanel = 0;              // A flag to initiate values of fullyOpened and fullyClosed timers
byte fullyOpenPanel = 0;                 // A flag that tells the panel is in fully opened position
byte fullyClosedPanel = 0;               // A flag that tells the panel is in fully closed position
byte bufferOpenPanel = 0;                // A one time buffer that allows the panel to open for one more program cycle when panel mov
ement is stopped either by user or hard stop
byte bufferClosePanel = 0;               // A one time buffer that allows the panel to close for one more program cycle when panel mo
vement is stopped either by user or hard stop
const int hardStopTimePanel = 100;       // Time in milliseconds for hard stop detection to trigger
const int fullyOpenDistancePanel = 1425; // The distance value that is forcefully assigned when panel is fully open
const int fullyClosedDistancePanel = 0;  // The distance value that is forcefully assigned when panel is fully closed

long Time_Between_Hs_Input_Panel = 0;    // The time between Hall Sensor inputs, gives information about how fast the panel is moving
 / it is the reverese of rpm
long Time_Between_Hs_Input_Panel_Old = 0;// Time between Hall Sensor Inputs in earlier iteration
long Time_Per_Revolution_Panel = 0;      // The time it takes to get 4 Hs inputs --> for half a rotation

int distancePanel = 0;                   // The number of Hs inputs recived from the starting position of the panel (fully closed)
long Tresh_base_pos = 0;                 // Parameter that enables the curve to adapt to the characteristics of the current operation
const int pinchSensitivityPanel = 0;     // Parameter that enables change of the amplitude of the treshold curve for all zones

// Treshold section parameters
const int UL1Panel = 956;                // Upper limit of the distance for Pinch Zone 1
const int LL1Panel = 650;                // Lower limit of the distance for Pinch Zone 1

const int UL2Panel = LL1Panel;           // Upper limit of the distance for Pinch Zone 2
const int LL2Panel = 520;                // Lower limit of the distance for Pinch Zone 2

const int UL3Panel = LL2Panel;           // Upper limit of the distance for Pinch Zone 3
const int LL3Panel = 380;                // Lower limit of the distance for Pinch Zone 3

const int UL4Panel = LL3Panel;           // Upper limit of the distance for Pinch Zone 4
const int LL4Panel = 280;                // Lower limit of the distance for Pinch Zone 4

const int UL5Panel = LL4Panel;           // Upper limit of the distance for Pinch Zone 5
const int LL5Panel = 205;                // Lower limit of the distance for Pinch Zone 5

const int UL6Panel = LL5Panel;           // Upper limit of the distance for Pinch Zone 6
const int LL6Panel = 150;                // Lower limit of the distance for Pinch Zone 6

const int UL7Panel = LL6Panel;           // Upper limit of the distance for Pinch Zone 7
const int LL7Panel = 90;                 // Lower limit of the distance for Pinch Zone 7

// Positive Treshold curve fit parameters on the form:  A* Distance^2 + B* Distance + C
// Zone 1
const float A1Panel = -0.0022;
const float B1Panel = 0.0318;
int C1Panel_tresh = 550;
int C1Panel = 4550;

// Zone 2
const float A2Panel = 0.1;
const float B2Panel = -3;
int C2Panel_tresh = 5850;    // C value for the adaptive Curve
int C2Panel = 9750;

// Zone 3
const float A3Panel = -0.08;
const float B3Panel = 7.0;
int C3Panel_tresh = 6710;   // C value for the adaptive Curve
int C3Panel = 10700;

// Zone 4
const float A4Panel = 0.07;
const float B4Panel = -3.31;
int C4Panel_tresh = 5700;  // C value for the adaptive Curve
int C4Panel = 9700;

// Zone 5
const float A5Panel = 0;
const float B5Panel = 0;
int C5Panel_tresh = 10050;// C value for the adaptive Curve
int C5Panel = 10100;
```

```
// Zone 6
const float A6Panel = 0;
const float B6Panel = 12.0;
//int C6Panel_tresh = 800;
int C6Panel_tresh = 1050;  // C value for the adaptive Curve
int C6Panel = 4900;

// Zone 7
const float A7Panel = 0;
const float B7Panel = -5.4011;
int C7Panel_tresh = 1950; // C value for the adaptive Curve
int C7Panel = 5500;



// Parameters used to set a state, varies between 1 & 0 depending on if they are on or off
int pinchDetectorPanel   = 0;         // Sets the system in "Pinch mode" 0 = Nothing is happening 1 = A pinch has occured

int stopInPinchZonePanel = 0;         // Allows the system to overwrite a Pinch if the user stops the pannel within the "Pinch Zone"
avoids fals pinches 0 = Nothing is happening 1 = Panel has stoped in the pinch zone
int stopDistancePanel    = 0;         // Sets the distance at which the panel has been stoped

//////////////////////////////////////////////////// Initiate Parameters Curtain /////////////////////////////////////////////////

volatile byte hsDetectCurtain  = 0;   // Hall Sensor detection in the Curtain
const byte hSensitivityCurtain = 1;   // How sensitive will the system be / the readings of the hall sensors
volatile byte confirmerCurtain = 0;   // Vairable used to help determine the distance

unsigned long hs1TimeCurtain = 0;     // Time when hall sensor 1 is detected
unsigned long hs2TimeCurtain = 0;     // Time when hall sensor 2 is detected

// Parameters for determining if the Curtain has reached its end position
long progTimeCurtain         = 0;     // A timer to start timing when the user presses the button
long fullyOpenedTimerCurtain = 0;     // A timer to measure time between the last hall sensor input and program time while closing th
e curtain
long fullyClosedTimerCurtain = 0;     // A timer to measure time between the last hall sensor input and program time while opening th
e curtain
byte oneTimeCheckCurtain     = 0;     // A flag to initiate values of fullyOpened and fullyClosed timers
byte fullyOpenCurtain        = 0;     // A flag that tells the curtain is in fully opened position
byte fullyClosedCurtain      = 0;     // A flag that tells the curtain is in fully closed position
byte bufferOpenCurtain       = 0;     // A one time buffer that allows the curtain to open for one more program cycle when curtain mo
vement is stopped either by user or hard stop
byte bufferCloseCurtain      = 0;     // A one time buffer that allows the curtain to close for one more program cycle when curtain m
ovement is stopped either by user or hard stop

const int hardStopTimeCurtain        = 100;   // Time in milliseconds for hard stop detection to trigger
const int fullyOpenDistanceCurtain   = 2150;  // The distance value that is forcefully assigned when curtain is fully open
const int fullyClosedDistanceCurtain = 0;     // The distance value that is forcefully assigned when curtain is fully closed

long Time_Between_Hs_Input_Curtain    = 0;    // The time between Hall Sensor inputs, gives information about how fast the Curtain is
 moving / it is the reverese of rpm
long Time_Between_Hs_Input_CurtainOld = 0;    // Time between Hall Sensor Inputs in earlier iteration
long Time_Per_Revolution_Curtain      = 0;    // The time it takes to get 4 Hs inputs --> for half a revolution
long Time_Per_Revolution_CurtainOld   = 0;    // Time per revolution in earlier iteration

int distanceCurtain         = 0;              // The number of Hs inputs recived from the starting position of the Curtain (fully clo
sed)
int distanceOldCurtain      = 0;              // The distance in earlier iteration
int SensitivityCurtain      = 0;              // Parameter that enables change of the amplitude of the treshold curve for all zones
long Tresh_base_Curtain     = 0;              // Parameter that enables the curve to adapt to the characteristics of the current oper
ation

// Upper & lowe limits for the Pinch Zones
const int UL1Curtain = 956;                   // Upper limit of the distance for Pinch Zone 1
const int LL1Curtain = 540;                   // Lower limit of the distance for Pinch Zone 1

const int UL2Curtain = LL1Curtain;            // Upper limit of the distance for Pinch Zone 2
const int LL2Curtain = 210;                   // Lower limit of the distance for Pinch Zone 2

const int UL3Curtain = LL2Curtain;            // Upper limit of the distance for Pinch Zone 3
const int LL3Curtain = 100;                   // Lower limit of the distance for Pinch Zone 3

const int UL4Curtain = LL3Curtain;            // Upper limit of the distance for Pinch Zone 4
const int LL4Curtain = 40;                    // Lower limit of the distance for Pinch Zone 4

// Treshold curve fit parameters on the form:  A* Distance^2 + B* Distance + C. Straight line for curtain so C is tha amplitude
// Zone 1
const int C1Curtain_tresh = 200;              // C value for the adaptive Curve
int C1Curtain             = 4700;

// Zone 2
const int C2Curtain_tresh = 300;              // C value for the adaptive Curve
int C2Curtain             = 4800;

// Zone 3
const int C3Curtain_tresh = 2300;             // C value for the adaptive Curve
int C3Curtain             = 6800;

// Zone 4
const int C4Curtain_tresh = 2400;             // C value for the adaptive Curve
int C4Curtain             = 6850;

// Parameters used to set a state, varies between 1 & 0 depending on if they are on or off
int pinchDetectorCurtain   = 0;               // Sets the system in "Pinch mode" 0 = Nothing is happening 1 = A pinch has occured

int stopInPinchZoneCurtain = 0;               // Allows the system to overwrite a Pinch if the user stops the Curtain within the "Pin
ch Zone" avoids fals pinches 0 = Nothing is happening 1 = Curtain has stoped in the pinch zone
int stopDistanceCurtain    = 0;               // Sets the distance at which the Curtain has been stoped
```

```
////////////////////////////////////////////////// Initiate Parameters Window ////////////////////////////////////////////////////

volatile byte hsDetectWindow = 0;          // Hall Sensor detection in the Window
volatile byte confirmerWindow = 0;         // Vairable used to help determine the distance
const byte hSensitivityWindow = 1;         // How sensitive will the system be / the readings of the hall sensors

unsigned long hs1TimeWindow = 0;           // Time when hall sensor 1 is detected
unsigned long hs2TimeWindow = 0;           // Time when hall sensor 2 is detected

// Parameters for determining if the Window has reached its end position
long progTimeWindow         = 0;           // A timer to start timing when the user presses the button
long fullyOpenedTimerWindow = 0;           // A timer to measure time between the last hall sensor input and program time while closi
ng the panel
long fullyClosedTimerWindow = 0;           // A timer to measure time between the last hall sensor input and program time while openi
ng the panel
byte oneTimeCheckWindow     = 0;           // A flag to initiate values of fullyOpened and fullyClosed timers
byte fullyOpenWindow        = 0;           // A flag that tells the panel is in fully opened position
byte fullyClosedWindow      = 0;           // A flag that tells the panel is in fully closed position
byte bufferOpenWindow       = 0;           // A one time buffer that allows the panel to open for one more program cycle when panel m
ovement is stopped either by user or hard stop
byte bufferCloseWindow      = 0;           // A one time buffer that allows the panel to close for one more program cycle when panel
movement is stopped either by user or hard stop
const byte hardStopTimeWindow      = 100; // Time in milliseconds for hard stop detection to trigger
const int fullyOpenDistanceWindow   = 455; // The distance value that is forcefully assigned when panel is fully open
const int fullyClosedDistanceWindow = 0;   // The distance value that is forcefully assigned when panel is fully closed

long Time_Between_Hs_Input_Window    = 0;  // The time between Hall Sensor inputs, gives information about how fast the Window is mov
ing / it is the reverese of rpm
long Time_Between_Hs_Input_WindowOld = 0;  // Time between Hall Sensor Inputs in earlier iteration
long Time_Per_Revolution_Window      = 0;  // The time it takes to get 4 Hs inputs --> for half a revolution

int distanceWindow       = 0;      // The number of Hs inputs recived from the starting position of the Window (fully closed)
long thresh_base_Window  = 0;      // Parameter that enables the curve to adapt to the characteristics of the current operation
int thresh_base_Window_5 = 18500; // Parameter that enables the curve to adapt to the characteristics of the current operation for zo
ne 5

const int pinchSensitivityWindow = 0;         // Used to lower or raise the curve against the pace input to change the sensitivity of
 the treshold curve fit

// Treshold section parameters
const int UL1Window = 310;                    // Upper limit of the distance for Pinch Zone 1
const int LL1Window = 270;                    // Lower limit of the distance for Pinch Zone 1

const int UL2Window = LL1Window;              // Upper limit of the distance for Pinch Zone 2
const int LL2Window = 230;                    // Lower limit of the distance for Pinch Zone 2

const int UL3Window = LL2Window;              // Upper limit of the distance for Pinch Zone 3
const int LL3Window = 150;                    // Lower limit of the distance for Pinch Zone 3

const int UL4Window = LL3Window;              // Upper limit of the distance for Pinch Zone 4
const int LL4Window = 100;                    // Lower limit of the distance for Pinch Zone 4

const int UL5Window = LL4Window;              // Upper limit of the distance for Pinch Zone 5
const int LL5Window = 50;                     // Lower limit of the distance for Pinch Zone 5

const int UL6Window = LL5Window;              // Upper limit of the distance for Pinch Zone 6
const int LL6Window = 15;                     // Lower limit of the distance for Pinch Zone 6



// Treshold curve fit parameters on the form:  A* Distance^2 + B* Distance + C for the different zones
// Zone 1
const float A1Window      = 0.04;
const float B1Window      = -2.69;
int C1Window              = 12000;
const int C1Window_thresh = 2700;             // C value for the adaptive Curve

// Zone 2
const float A2Window      = 0.04;
const float B2Window      = -2.69;
int C2Window              = 13000;
const int C2Window_thresh = 3000;             // C value for the adaptive Curve

// Zone 3
const float A3Window      = 0.04;
const float B3Window      = -1.0;
int C3Window              = 14000;
const int C3Window_thresh = 3800;             // C value for the adaptive Curve

// Zone 4
const float A4Window      = -0.04;
const float B4Window      = -1;
int C4Window              = 17000;
const int C4Window_thresh = 7000;             // C value for the adaptive Curve

// Zone 5
const float A5Window        = -3.04;
const int B5Window          = 223;
unsigned int C5Window       = 30000;
const int C5Window_thresh   = 24000;          // C value for the adaptive Curve
const int C5Window_thresh_5 = 8500;           // C value for the adaptive Curve only zone 5

// Zone 6
const int A6Window          = 0;
const int B6Window          = 0;
unsigned long C6Window      = 20500;          // C value for the adaptive Curve
```

```
const int C6Window_thresh   = 15000;          // C value for the adaptive Curve only zone 6

// Parameters used to set a state, varies between 1 & 0 depending on if they are on or off
int pinchDetectorWindow     = 0;              // Sets the system in "Pinch mode" 0 = Nothing is happening 1 = A pinch has occured
int stopInPinchZoneWindow   = 0;              // Allows the system to overwrite a Pinch if the user stops the window within the "Pinc
h Zone" avoids fals pinches 0 = Nothing is happening 1 = Window has stoped in the pinch zone
int stopDistanceWindow      = 0;              // Sets the distance at which the Window has been stoped

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// Defining input/output pins
////////////////////////// Input Panel
////////////////////////// Input/Output pins Panel
const int closeButtonPanel   = 6;             // Defines the pin on the Arduino for the button that closes the panel
const int openButtonPanel    = 7;             // Defines the pin on the Arduino for the button that opens the panel
const int hs1Panel           = 20;            // Defines the pin on the Arduino which recives the input from hall sensor 1 for the pa
nel
const int hs2Panel           = 21;            // Defines the pin on the Arduino which recives the input from hall sensor 2 for the pa
nel

//////////////////////// Input/Output pins Curtain
const int closeButtonCurtain = 4;             // Defines the pin on the Arduino for the button that closes the curtain
const int openButtonCurtain  = 5;             // Defines the pin on the Arduino for the button that opens the curtain
const int hs1Curtain         = 18;            // Defines the pin on the Arduino which recives the input from hall sensor 1 for the cu
rtain
const int hs2Curtain         = 19;            // Defines the pin on the Arduino which recives the input from hall sensor 2 for the cu
rtain

////////////////////////// Input/Output pins Window
const int closeButtonWindow  = 4;             // Defines the pin on the Arduino for the button that closes the window
const int openButtonWindow   = 5;             // Defines the pin on the Arduino for the button that opens the window
const int hs1Window          = 2;             // Defines the pin on the Arduino which recives the input from hall sensor 1 for the wi
ndow (only one hs for window)
const byte curtainVSwindow   = 3;             // Defines the pin on the Arduino which says if the switch is active for the window or
the curtain

////////////////////// Speed Control

const byte nSLEEP = 7;
const byte VCC    = 8;

const byte PHpinPanel   = 9;                  // Corresponds to direction of rotation (OCR2B)
const byte ENpinPanel   = 10;                 // Corresponds to PulsWdtMod duty cycle (OCR2A)
const byte PHpinCurtain = 11;                 // Corresponds to direction of rotation (OCR1A)
const byte ENpinCurtain = 12;                 // COrresponds to PulsWdtMod duty cycle (OCR1B)

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////// All parameters and variables defined ////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup() // Void Setup
{
  Serial.begin(115200);

//////////////////////////// Setup direction & Speed Control

  // PWM control changing frequency for TIMER 2
  TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM20);
  TCCR2B = TCCR2B & 0b11111000 | 0x01;

  // PWM control changing frequency for TIMER 1
  TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM20);
  TCCR1B = TCCR1B & 0b11111000 | 0x01;
  pinMode(nSLEEP, OUTPUT);
  pinMode(VCC, OUTPUT);
  pinMode(PHpinPanel, OUTPUT);
  pinMode(ENpinPanel, OUTPUT);
  pinMode(PHpinCurtain, OUTPUT);
  pinMode(ENpinCurtain, OUTPUT);
  digitalWrite(nSLEEP, HIGH);
  digitalWrite(VCC, HIGH);

  ///////////////////////// Void Setup Panel

  pinMode(openButtonPanel, INPUT);        // Open Button is an input
  pinMode(closeButtonPanel, INPUT);       // Close Button is an input
  OCR2A = 0;                              // Initial state of Direction & speed is 0 = nothing happens before button is pressed
  OCR2B = 0;                              // Initial state of Direction & speed is 0 = nothing happens before button is pressed

  attachInterrupt(digitalPinToInterrupt(hs1Panel), magnet_detect1Panel, RISING); // Interuppts the system if a hall sensor is detecte
  attachInterrupt(digitalPinToInterrupt(hs2Panel), magnet_detect2Panel, RISING); // Interuppts the system if a hall sensor is detecte

  ///////////////////////// Setup Curtain

  pinMode(openButtonCurtain, INPUT);      // Open Button is an input
  pinMode(closeButtonCurtain, INPUT);     // Close Button is an input
  OCR1A = 0;                              // Initial state of Direction & speed is 0 = nothing happens before button is pressed
  OCR1B = 0;                              // Initial state of Direction & speed is 0 = nothing happens before button is pressed

  attachInterrupt(digitalPinToInterrupt(hs1Curtain), magnet_detect1Curtain, RISING); // Interuppts the system if a hall sensor is det
ected
  attachInterrupt(digitalPinToInterrupt(hs2Curtain), magnet_detect2Curtain, RISING); // Interuppts the system if a hall sensor is det
ected

  ///////////////////////// Setup Window

  pinMode(openButtonWindow, INPUT);       // Open Button is an input
  pinMode(closeButtonWindow, INPUT);      // Close Button is an input
  pinMode(curtainVSwindow, INPUT);
  attachInterrupt(digitalPinToInterrupt(hs1Window), magnet_detect1Window, RISING); // Interuppts the system if a hall sensor is detec
ted (only one for window)

} // Ends the void setup

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// PANEL
////////////////////////// OPEN Panel
void loop()
{

// Enter the open statement if the open buttons is presses and the panel is not in its fully open position
  if (digitalRead(openButtonPanel) == HIGH && fullyOpenPanel == 0)
  {
    fullyClosedPanel = 0;               // Resets the "fullyClosedPanel" flag to 0
    stopInPinchZonePanel = 0;           // Resets the "stopInPinchZonePanel" flag to 0

     if (distancePanel < 300)
    {
    OCR2A = 255;                        // Speed = 100%
    OCR2B = 0;                          // Opening direction
    }
    else if (distancePanel > 1300)
    {
    OCR2A = 255;                        // Speed = 100%
    OCR2B = 0;                          // Opening direction
    }
    else
    {
    OCR2A = 235;                        // Speed = 90%
    OCR2B = 0;                          // Opening direction
    }
    DistanceOpenFunkPanel();            // Calls the Distance open function
  }                                     // Ends the opening statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CLOSE Panel

// Enter the close statement if the close buttons is presses, the panel is not in its fully closed position,
//  if there is no stop in the pinch zone, and if no pinch is detected

  else if (digitalRead(closeButtonPanel) == HIGH && stopInPinchZonePanel == 0 && pinchDetectorPanel == 0 && fullyClosedPanel == 0)
  {
    fullyOpenPanel = 0;                 // Resets the "fullyOpenPanel" flag to 0

    if (distancePanel > UL1Panel + 4)
    {
    OCR2A = 255;                        // Speed = 100%
    OCR2B = 255;                        // Closing direction

    }
    else if (distancePanel > UL1Panel && distancePanel <= UL1Panel + 4)
    {
    OCR2A = 255;                        // Speed = 100%
    OCR2B = 255;                        // Closing direction

      if (Time_Per_Revolution_Panel > Tresh_base_pos)// If The time per revolution is higher than the treshold base will a new tresho
ld value be given
      {
        Tresh_base_pos = Time_Per_Revolution_Panel;  // A new base value is assigned to the treshholdcurve that is unique for this op
eration
      }

      // Asigned the Adaptive treshold base value to the C values for the curve
      C1Panel = Tresh_base_pos + C1Panel_tresh;           // Zone 1
      C2Panel = Tresh_base_pos + C2Panel_tresh;           // Zone 2
      C3Panel = Tresh_base_pos + C3Panel_tresh;           // Zone 3
      C4Panel = Tresh_base_pos + C4Panel_tresh;           // Zone 4
      C5Panel = Tresh_base_pos + C5Panel_tresh;           // Zone 5
      C6Panel = Tresh_base_pos + C6Panel_tresh;           // Zone 6
      C7Panel = Tresh_base_pos + C7Panel_tresh;           // Zone 7
    }

    else if (distancePanel >= LL1Panel && distancePanel < UL1Panel) // Enters Pinch Zone 1
    {
    OCR2A = 235;                    // Speed = 90%
    OCR2B = 255;                    // Closing direction

      // Checks if the treshold curve for Zone 1 is lower than the "time per revolution"
      if (A1Panel * (UL1Panel - distancePanel) * (UL1Panel - distancePanel) + B1Panel * (UL1Panel - distancePanel) + C1Panel - pinchS
ensitivityPanel < Time_Per_Revolution_Panel)
      {
        pinchDetectorPanel = 1;     // If the treshold curve is lower than "time per revolution" is set to pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A1Panel * (UL1Panel - distancePanel) * (UL1Panel - distancePanel) + B1Panel * (UL1Panel - distancePanel) +
 C1Panel - pinchSensitivityPanel));
        Serial.print("; ");
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.print("Zone1");
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
      }
    }

    else if (distancePanel >= LL2Panel && distancePanel < UL2Panel)  // Enters Pinch Zone 2
    {
    OCR2A = 220;                    // Speed = 86%
    OCR2B = 255;                    // Closing direction

      // Checks if the treshold curve for Zone 2 is lower than the "time per revolution"
      if (distancePanel >= LL2Panel - 5  && A2Panel * (UL2Panel - distancePanel) * (UL2Panel - distancePanel) + B2Panel * (UL2Panel -
 distancePanel) + C2Panel - pinchSensitivityPanel < Time_Per_Revolution_Panel)
      {
        pinchDetectorPanel = 1;     // If the treshold curve is lower than "time per revolution" is set to pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A2Panel * (UL2Panel - distancePanel) * (UL2Panel - distancePanel) + B2Panel * (UL2Panel - distancePanel) +
 C2Panel - pinchSensitivityPanel));
        Serial.print("; ");
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.print("Zone2");
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
      }
    }

    else if (distancePanel >= LL3Panel && distancePanel < UL3Panel) // Enters Pinch Zone 3
    {
    OCR2A = 225;                    // Speed = 88%
    OCR2B = 255;                    // Closing direction

      // Checks if the treshold curve for Zone 3 is lower than the "time per revolution"
      if (distancePanel < UL3Panel - 5 && A3Panel * (UL3Panel - distancePanel) * (UL3Panel - distancePanel) + B3Panel * (UL3Panel - d
istancePanel) + C3Panel - pinchSensitivityPanel < Time_Per_Revolution_Panel)
      {
        pinchDetectorPanel = 1;       // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A3Panel * (UL3Panel - distancePanel) * (UL3Panel - distancePanel) + B3Panel * (UL3Panel - distancePanel) +
 C3Panel - pinchSensitivityPanel));
        Serial.print("; ");
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.print("Zone3");
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
      }
    }

    else if (distancePanel >= LL4Panel && distancePanel < UL4Panel) // Enters Pinch Zone 4
    {

    OCR2A = 226;                    // Speed = 89%
    OCR2B = 255;                    // Closing direction

      // Checks if the treshold curve for Zone 4 is lower than the "time per revolution"
      if (A4Panel * (UL4Panel - distancePanel) * (UL4Panel - distancePanel) + B4Panel * (UL4Panel - distancePanel) + C4Panel - pinchS
ensitivityPanel < Time_Per_Revolution_Panel)
      {
        pinchDetectorPanel = 1;  // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A4Panel * (UL4Panel - distancePanel) * (UL4Panel - distancePanel) + B4Panel * (UL4Panel - distancePanel) +
 C4Panel - pinchSensitivityPanel));
        Serial.print("; ");
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.print("Zone4");
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
      }
    }

    else if (distancePanel >= LL5Panel && distancePanel < UL5Panel)  // Enters Pinch Zone 5
    {

    OCR2A = 226;                    // Speed = 89%
    OCR2B = 255;                    // Closing direction

      // Checks if the treshold curve for Zone 5 is lower than the "time per revolution"
      if (A5Panel * (UL5Panel - distancePanel) * (UL5Panel - distancePanel) + B5Panel * (UL5Panel - distancePanel) + C5Panel - pinchS
ensitivityPanel < Time_Per_Revolution_Panel)
      {
        pinchDetectorPanel = 1;    // If the treshold curve is lower than "time per revolution" --> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A5Panel * (UL5Panel - distancePanel) * (UL5Panel - distancePanel) + B5Panel * (UL5Panel - distancePanel) +
 C5Panel - pinchSensitivityPanel));
        Serial.print("; ");
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.print("Zone5");
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
      }
    }

    else if (distancePanel >= LL6Panel && distancePanel < UL6Panel)       // Enters Pinch Zone 6
    {
    OCR2A = 240;                    // Speed = 94%
    OCR2B = 255;                    // Closing direction

      if (distancePanel < UL6Panel - 10 && A6Panel * (UL6Panel - distancePanel) * (UL6Panel - distancePanel) + B6Panel * (UL6Panel -
distancePanel) + C6Panel - pinchSensitivityPanel < Time_Per_Revolution_Panel)
      {
        pinchDetectorPanel = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A6Panel * (UL6Panel - distancePanel) * (UL6Panel - distancePanel) + B6Panel * (UL6Panel - distancePanel) +
 C6Panel - pinchSensitivityPanel));
        Serial.print("; ");
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.print("Zone6");
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
      }
    }

    else if (distancePanel >= LL7Panel && distancePanel < UL7Panel) // Enters Pinch Zone 7
    {
    OCR2A = 240;                    // Speed = 94%
    OCR2B = 255;                    // Closing direction

      // Checks if the treshold curve for Zone 7 is lower than the "time per revolution"
      if (A7Panel * (UL7Panel - distancePanel) * (UL7Panel - distancePanel) + B7Panel * (UL7Panel - distancePanel) + C7Panel - pinchS
ensitivityPanel < Time_Per_Revolution_Panel)
      {
        pinchDetectorPanel = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A7Panel * (UL7Panel - distancePanel) * (UL7Panel - distancePanel) + B7Panel * (UL7Panel - distancePanel) +
 C7Panel - pinchSensitivityPanel));
        Serial.print("; ");
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.print("Zone7");
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
      }
    }
    else if (distancePanel < LL7Panel )
    {
    OCR2A = 255;                    // Speed = 100%
    OCR2B = 255;                    // Closing direction
    }

    if (pinchDetectorPanel == 0)    // enters if no pinch is detected
    {
      DistanceCloseFunkPanel();     // Calls the Distance close function
    }
  } //          Stops Close statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

  else if (pinchDetectorPanel == 1) // If a pinch is detected in the pannel will it change direction and open fully
  {
    OCR2A = 255;                        // Speed 100%
    OCR2B = 0;                          // Opening direction
    DistanceOpenFunkPanel();            // Cals the Distance opening function
  }                                     // Stops pinch in panel statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///// Avoids falls pinches by overwriting the pinch stement if the user stops the panel within the pinch zone
// Enters if Close button is high, stop in pinch zone is high and the pannel is not fully closed

  else if (digitalRead(closeButtonPanel) == HIGH && stopInPinchZonePanel == 1 && fullyClosedPanel == 0)
  {

    OCR2A = 255;                        // Speed 100%
    OCR2B = 255;                        // Closing direction
    DistanceCloseFunkPanel();           // Cals the Distance closing function
  }                                     // Stops Stop in pinch zone statement

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////// Allow the user to stop the panel within the pinch zone
// Enters if both buttons are low and the panel is in the pinch zone
  else if (digitalRead(openButtonPanel) == LOW && digitalRead(closeButtonPanel) == LOW && distancePanel > LL7Panel && distancePanel <
 UL1Panel)
  {
    fullyOpenPanel = 0;                 // Resets the "fullyOpenPanel" flag to 0
    fullyClosedPanel = 0;               // Resets the "fullyClosedPanel" flag to 0

    OCR2A = 0;                          // Speed 0%
    OCR2B = 0;                          // Opening direction/ doesn't matter which direction
    stopInPinchZonePanel = 1;           // Sets the "stopInPinchZonePanel" flag to 1
    stopDistancePanel = distancePanel;  // Assigns the distance value at which the pannel is stoped
  }

  else    // If nothing happens
  {

    OCR2A = 0;                          // Speed 0%
    OCR2B = 0;                          // Opening direction/ doesn't matter which direction
  }
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// OPEN Curtain ///
if (digitalRead(curtainVSwindow) == HIGH)   // If switch is in curtain position(HIGH)
{
// Enter the open statement if the open buttons is presses and the curtain is not in its fully open position
if (digitalRead(openButtonCurtain) == HIGH && fullyOpenCurtain == 0)
{
  fullyClosedCurtain = 0;           // Resets the "fullyClosedCurtain" flag to 0
  OCR1A = 0;                        // Opening Direction
  OCR1B = 255;                      // Speed 100% PWM
  DistanceOpenFunkCurtain();        // Calls the Distance open function
}                                   // Ends the opening statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CLOSE Curtain

// Enter the close statement if the close buttons is presses, the curtain is not in its fully closed position,
// if there is no stop in the pinch zone, and if no pinch is detected

  else if (digitalRead(closeButtonCurtain) == HIGH && stopInPinchZoneCurtain == 0 && pinchDetectorCurtain == 0 && fullyClosedCurtain
== 0)
  {

  fullyOpenCurtain = 0;             // Resets the "fullyOpenedCurtain" flag to 0

    if (distanceCurtain > UL1Curtain + 4)
    {
  OCR1A = 255;                      // Closing Direction
  OCR1B = 255;                      // Speed 100% PWM
    }
    else if (distanceCurtain > UL1Curtain && distanceCurtain <= UL1Curtain + 4)
    {
  OCR1A = 255;                      // Closing Direction
  OCR1B = 255;                      // Speed 100% PWM



    if (Time_Per_Revolution_Curtain > Tresh_base_Curtain)  // If The time per revolution is higher than the treshold base will a new
treshold value be given
    {
      Tresh_base_Curtain = Time_Per_Revolution_Curtain;    // A new base value is assigned to the treshholdcurve that is unique for t
his operation
    }
    // Asigned the Adaptive treshold base value to the C values for the curve
    C1Curtain = Tresh_base_Curtain + C1Curtain_tresh;     // Zone 1
    C2Curtain = Tresh_base_Curtain + C2Curtain_tresh;     // Zone 2
    C3Curtain = Tresh_base_Curtain + C3Curtain_tresh;     // Zone 3
    C4Curtain = Tresh_base_Curtain + C4Curtain_tresh;     // Zone 4
  }

    else if (distanceCurtain >= LL1Curtain && distanceCurtain < UL1Curtain) // Enters Pinch Zone 1
    {
  OCR1A = 255;                      // Closing Direction
  OCR1B = 255;                      // Speed 100% PWM

    // Checks if the treshold curve for Zone 1 is lower than the "time per revolution"
      if (C1Curtain < Time_Per_Revolution_Curtain )
      {
      pinchDetectorCurtain = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

      // Printind details about the pinch situation
      Serial.print(distanceCurtain);
      Serial.print("; ");
      Serial.print(Time_Per_Revolution_Curtain);
      Serial.print("; ");
      Serial.print("Zone1");
      Serial.print("; ");
      Serial.println(C1Curtain);
      }
    }
    else if (distanceCurtain >= LL2Curtain && distanceCurtain < UL2Curtain)  // Enters Pinch Zone 2
    {
  OCR1A = 255;                      // Closing Direction
  OCR1B = 255;                      // Speed 100% PWM

      // Checks if the treshold curve for Zone 2 is lower than the "time per revolution"
      if (C2Curtain < Time_Per_Revolution_Curtain )
      {
      pinchDetectorCurtain = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

      // Printind details about the pinch situation
      Serial.print(distanceCurtain);
      Serial.print("; ");
      Serial.print(Time_Per_Revolution_Curtain);
      Serial.print("; ");
      Serial.print("Zone2 ");
      Serial.print("; ");
      Serial.println(C2Curtain);
      }
    }
    else if (distanceCurtain >= LL3Curtain && distanceCurtain < UL3Curtain)     // Enters Pinch Zone 3
    {
  OCR1A = 255;                      // Closing Direction
  OCR1B = 200;                      // Speed 78% PWM

      if (C3Curtain < Time_Per_Revolution_Curtain )
      {
      pinchDetectorCurtain = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

      // Printind details about the pinch situation
      Serial.print(distanceCurtain);
      Serial.print("; ");
      Serial.print(Time_Per_Revolution_Curtain);
      Serial.print("; ");
      Serial.print("Zone3 ");
      Serial.print("; ");
      Serial.println(C3Curtain);
      }
    }
    else if (distanceCurtain >= LL4Curtain && distanceCurtain < UL4Curtain)     // Enters Pinch Zone 4
    {
  OCR1A = 255;                      // Closing Direction
  OCR1B = 200;                      // Speed 78% PWM

    // Checks if the treshold curve for Zone 4 is lower than the "time per revolution"
      if (C4Curtain < Time_Per_Revolution_Curtain )
      {
      pinchDetectorCurtain = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

      // Printind details about the pinch situation
      Serial.print(distanceCurtain);
      Serial.print("; ");
      Serial.print(Time_Per_Revolution_Curtain);
      Serial.print("; ");
      Serial.print("Zone4 ");
      Serial.print("; ");
      Serial.println(C4Curtain);
      }
    }
    else if (distanceCurtain < LL4Curtain)
    {
  OCR1A = 255;                      // Closing Direction
  OCR1B = 255;                      // Speed 100% PWM
    }

  if (pinchDetectorCurtain == 0)    // enters if no pinch is detected
  {
    DistanceCloseFunkCurtain();     // Calls the Distance close function
  }
  } // Stops closing of curtain

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
else if (pinchDetectorCurtain == 1)// If a pinch is detected in the curtain will it change direction and open fully
{

  fullyOpenCurtain = 0;                   // Resets the "fullyOpenCurtain" flag to 0
  OCR1A = 0;                              // Opening Direction
  OCR1B = 255;                            // Speed 100% PWM
  DistanceOpenFunkCurtain();              // Cals the Distance opening function

}                                         // Stops Pinch in curtain statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///// Avoids falls pinches by overwriting the pinch stement if the user stops the curtain within the pinch zone
// Enters if Close button is high, stop in pinch zone is high and the curtain is not fully closed

else if (digitalRead(closeButtonCurtain) == HIGH && stopInPinchZoneCurtain == 1 && fullyClosedCurtain == 0)
{
  fullyOpenCurtain = 0;                   // Resets the "fullyOpenCurtain" flag to 0
  OCR1A = 255;                            // Closing Direction
  OCR1B = 255;                            // Speed 100% PWM
  DistanceCloseFunkCurtain();             // Cals the Distance close function
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////// Allow the user to stop the curtain within the pinch zone
// Enters if both buttons are low and the curtain is in the pinch zone

else if (digitalRead(openButtonCurtain) == LOW && digitalRead(closeButtonCurtain) == LOW && distanceCurtain > LL4Curtain && distanceC
urtain < UL1Curtain)
{
  OCR1A = 255;                            // Closing direction/ doesn't matter which direction
  OCR1B = 0;                              // Speed 0% PWM
  stopInPinchZoneCurtain = 1;             // Sets the "stopInPinchZoneCurtain" flag to 1
  stopDistanceCurtain = distanceCurtain;  // Assigns the distance value at which the curtain is stoped
}                                         // Stops Stop in pinch zone statement



else                                      // If nothing happens in Curtain
{
  OCR1A = 255;                            // Closing direction/ doesn't matter which direction
  OCR1B = 0;                              // Speed 0% PWM
}                                         // Stops if nothing happens statement
}                                         // Stops switch statement
///////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////// OPEN Window //////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////
else if (digitalRead(curtainVSwindow) == LOW)   // If switch is in Window position(LOW)
{

 // Enter the open statement if the open buttons is presses and the window is not in its fully open position
  if (digitalRead(openButtonWindow) == HIGH && fullyOpenWindow == 0)
  {
    fullyClosedWindow = 0;                // Resets the "fullyClosedWindow" flag to 0

    if (distanceWindow < LL5Window)
    {
      OCR1A = 0;                            // Opening Direction
      OCR1B = 180;                          // Speed 50% PWM
    }
    else if (distanceWindow >= LL5Window && distanceWindow < LL4Window)
    {
      OCR1A = 0;                            // Opening Direction
      OCR1B = 200;                          // Speed 78% PWM
    }
    else if (distanceWindow >= LL4Window && distanceWindow < LL3Window)
    {
      OCR1A = 0;                            // Opening Direction
      OCR1B = 220;                          // Speed 86% PWM
    }
    else if (distanceWindow >= LL3Window && distanceWindow < LL2Window)
    {
      OCR1A = 0;                            // Opening Direction
      OCR1B = 240;                          // Speed 94% PWM
    }
    else if (distanceWindow > 450 && distanceWindow < 500)
    {
      OCR1A = 0;                            // Opening Direction
      OCR1B = 180;                          // Speed 50% PWM
    }
    else if (distanceWindow >= 500)
    {
      OCR1A = 0;                            // Opening Direction
      OCR1B = 255;                          // Speed 100% PWM
    }
    else
    {
      OCR1A = 0;                            // Opening Direction
      OCR1B = 255;                          // Speed 100% PWM
    }
    DistanceOpenFunkWindow();               // Calls the Distance open function
  }

///////////////////////////////////////////////////////////////////////////////////////////////
// CLOSE Window

// Enter the close statement if the close buttons is presses, the window is not in its fully closed position,
// if there is no stop in the pinch zone, and if no pinch is detected

  else if (digitalRead(closeButtonWindow) == HIGH && stopInPinchZoneWindow == 0 && pinchDetectorWindow == 0 && fullyClosedWindow == 0
)
  {
    fullyOpenWindow = 0;                    // Resets the "fullyOpenWindow" flag to 0
    if (distanceWindow > UL1Window + 4)
    {
    OCR1A = 255;                            // Closing Direction
    OCR1B = 255;                            // Speed 100% PWM
    }
    else if (distanceWindow > UL1Window && distanceWindow <= UL1Window + 4)     // Adaptive threshold curve
    {
    OCR1A = 255;                            // Closing Direction
    OCR1B = 255;                            // Speed 100% PWM

      // If The time per revolution is higher than the treshold base will a new treshold value be given
      if (Time_Per_Revolution_Window  > thresh_base_Window)
      {
        // A new base value is assigned to the treshholdcurve that is unique for this operation
        thresh_base_Window = Time_Per_Revolution_Window ;
      }
      // Asigned the Adaptive treshold base value to the C values for the curve
      C1Window = thresh_base_Window + C1Window_thresh;     // Zone 1
      C2Window = thresh_base_Window + C2Window_thresh;     // Zone 2
      C3Window = thresh_base_Window + C3Window_thresh;     // Zone 3
      C4Window = thresh_base_Window + C4Window_thresh;     // Zone 4
      C5Window = thresh_base_Window + C5Window_thresh;     // Zone 5
      C6Window = thresh_base_Window + C6Window_thresh;     // Zone 6
    }
    else if (distanceWindow >= LL1Window && distanceWindow < UL1Window)     // Enters Pinch Zone 1
    {
    OCR1A = 255;                            // Closing Direction
    OCR1B = 255;                            // Speed 100% PWM

      // Checks if the treshold curve for Zone 1 is lower than the "time per revolution"
      if (A1Window * (UL1Window - distanceWindow) * (UL1Window - distanceWindow) + B1Window * (UL1Window - distanceWindow) + C1Window
 - pinchSensitivityWindow < Time_Per_Revolution_Window )
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A1Window * (UL1Window - distanceWindow) * (UL1Window - distanceWindow) + B1Window * (UL1Window - distanceW
indow) + C1Window - pinchSensitivityWindow));
```

```
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.print("Zone1");
      }
    }
    else if (distanceWindow >= LL2Window && distanceWindow < UL2Window)     // Enters Pinch Zone 2
    {
      OCR1A = 255;                            // Closing Direction
      OCR1B = 240;                            // Speed 94% PWM

      // Checks if the treshold curve for Zone 2 is lower than the "time per revolution"
      if ( A2Window * (UL2Window - distanceWindow) * (UL2Window - distanceWindow) + B2Window * (UL2Window - distanceWindow) + C2Window - pinchSensitivityWindow < Time_Per_Revolution_Window )
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A2Window * (UL2Window - distanceWindow) * (UL2Window - distanceWindow) + B2Window * (UL2Window - distanceWindow) + C2Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone2");

      }
    }
    else if (distanceWindow >= LL3Window && distanceWindow < UL3Window)     // Enters Pinch Zone 3
    {
      OCR1A = 255;                            // Closing Direction
      OCR1B = 230;                            // Speed 90% PWM

      // Checks if the treshold curve for Zone 3 is lower than the "time per revolution"
      if (A3Window * (UL3Window - distanceWindow) * (UL3Window - distanceWindow) + B3Window * (UL3Window - distanceWindow) + C3Window - pinchSensitivityWindow < Time_Per_Revolution_Window )
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A3Window * (UL3Window - distanceWindow) * (UL3Window - distanceWindow) + B3Window * (UL3Window - distanceWindow) + C3Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone3");

      }
    }
    else if (distanceWindow >= LL4Window && distanceWindow < UL4Window)  // Enters Pinch Zone 4
    {
      OCR1A = 255;                            // Closing Direction
      OCR1B = 210;                            // Speed 82% PWM

      // Checks if the treshold curve for Zone 4 is lower than the "time per revolution"
      if (A4Window * (UL4Window - distanceWindow) * (UL4Window - distanceWindow) + B4Window * (UL4Window - distanceWindow) + C4Window - pinchSensitivityWindow < Time_Per_Revolution_Window )
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A4Window * (UL4Window - distanceWindow) * (UL4Window - distanceWindow) + B4Window * (UL4Window - distanceWindow) + C4Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone4");

      }
    }
    else if (distanceWindow >= LL5Window && distanceWindow < UL5Window)     // Enters Pinch Zone 5
    {
      OCR1A = 255;                            // Closing Direction
      OCR1B = 150;                            // Speed 58% PWM

        // New adaptive c value due to very low speed
      if (distanceWindow >= UL5Window - 7 && distanceWindow < UL5Window-4)
      {
        if (Time_Per_Revolution_Window  > thresh_base_Window_5)
      {
        thresh_base_Window_5 = Time_Per_Revolution_Window ;
      }

      C5Window = thresh_base_Window_5 + C5Window_thresh_5;
      }

      // Checks if the treshold curve for Zone 5 is lower than the "time per revolution"
      if (distanceWindow >= LL5Window - 4 && A5Window * (UL5Window - distanceWindow) * (UL5Window - distanceWindow) + B5Window * (UL5Window - distanceWindow) + C5Window - pinchSensitivityWindow < Time_Per_Revolution_Window )
      {
        pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A5Window * (UL5Window - distanceWindow) * (UL5Window - distanceWindow) + B5Window * (UL5Window - distanceW
indow) + C5Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone5");

      }
    }
    else if (distanceWindow >= LL6Window && distanceWindow < UL6Window)    // Enters Pinch Zone 6
    {

      // Checks if the treshold curve for Zone 6 is lower than the "time per revolution"
      OCR1A = 255;                            // Closing Direction
      OCR1B = 180;                            // Speed 70% PWM
      if (distanceWindow < UL6Window - 4 && A6Window * (UL6Window - distanceWindow) * (UL6Window - distanceWindow) + B6Window * (UL6W
indow - distanceWindow) + C6Window - pinchSensitivityWindow < Time_Per_Revolution_Window )
      {
         pinchDetectorWindow = 1;     // If the treshold curve is lower than "time per revolution" is a pinch detected --
> pinchdetector = 1

        // Printind details about the pinch situation
        Serial.print(round(A6Window * (UL6Window - distanceWindow) * (UL6Window - distanceWindow) + B6Window * (UL6Window - distanceW
indow) + C6Window - pinchSensitivityWindow));
        Serial.print("; ");
        Serial.print(distanceWindow);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Window);
        Serial.print("; ");
        Serial.println("Zone6");
        Serial.print("; ");
        Serial.println(C6Window);
      }
    }
    else if (distanceWindow <= LL6Window)
    {
      OCR1A = 255;                          // Closing Direction
      OCR1B = 200;                          // Speed 78% PWM
    }
    if (pinchDetectorWindow == 0)           // enters if no pinch is detected
    {
      DistanceCloseFunkWindow();            // Calls the Distance close function
    }
  } // Stops Close statement

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
  else if (pinchDetectorWindow == 1)// If a pinch is detected in the window will it change direction and open fully
  {

    fullyOpenWindow = 0;             // Resets the "fullyOpenWindow" flag to 0
    OCR1A = 0;                       // Opening Direction
    OCR1B = 255;                     // Speed 100% PWM
    DistanceOpenFunkWindow();        // Cals the Distance opening function
  }                                  // Stops pinch in Window statement

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///// Avoids falls pinches by overwriting the pinch stement if the user stops the window within the pinch zone
// Enters if Close button is high, stop in pinch zone is high and the window is not fully closed

  else if (digitalRead(closeButtonWindow) == HIGH && stopInPinchZoneWindow == 1 && fullyClosedWindow == 0)
  {
    fullyOpenWindow = 0;             // Resets the "fullyOpenWindow" flag to 0
    OCR1A = 255;                     // Closing Direction
    OCR1B = 240;                     // Speed 94% PWM
    DistanceCloseFunkWindow();       // Cals the Distance close function
  }                                  // Stops Stop in pinch zone statement

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////// Allow the user to stop the window within the pinch zone
// Enters if both buttons are low and the window is in the pinch zone

  else if (digitalRead(openButtonWindow) == LOW && digitalRead(closeButtonWindow) == LOW  && distanceWindow > LL6Window && distanceWi
ndow < UL1Window)
  {

    OCR1A = 255;                            // Closing direction/ doesn't matter which direction
    OCR1B = 0;                              // Speed 0% PWM
    stopInPinchZoneWindow = 1;              // Sets the "stopInPinchZoneWindow" flag to 1
    stopDistanceWindow = distanceWindow;    // Assigns the distance value at which the window is stoped
  }                                         // Stops Stop in pinch zone statement

  else                                      // If nothing happens in Window
  {
    OCR1A = 255;                            // Closing direction/ doesn't matter which direction
    OCR1B = 0;                              // Speed 0% PWM
  }                                         // Stops if nothing happens statement

}
} //ENDS  VOID LOOP
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculate the pacing in Panel
void PaceFunkPanel()
```

```
{
  Time_Between_Hs_Input_Panel = hs1TimePanel - hs2TimePanel;           // Inverse of RPM
  Time_Between_Hs_Input_Panel = abs(Time_Between_Hs_Input_Panel);      // Take the absolute value

  Time_Per_Revolution_Panel   = Time_Between_Hs_Input_Panel - Time_Between_Hs_Input_Panel_Old;   // Checks the difference between the
 old and new time difference between hs input
  Time_Per_Revolution_Panel   = abs(Time_Per_Revolution_Panel);        // Take the absolute value

  hsDetectPanel = 0;                                                   // Resets Hs detect to 0

  Time_Between_Hs_Input_Panel_Old  = Time_Between_Hs_Input_Panel;      // Updated the Time_Between_Hs_Input_Panel_Old

// Printing of the treshold curve
  if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL1Panel && distancePanel < UL1Panel)
  {
    Serial.print(round(A1Panel * (UL1Panel - distancePanel) * (UL1Panel - distancePanel) + B1Panel * (UL1Panel - distancePanel) + C1P
anel - pinchSensitivityPanel));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL2Panel - 5 && distancePanel < UL2Panel)
  {
    Serial.print(round(A2Panel * (UL2Panel - distancePanel) * (UL2Panel - distancePanel) + B2Panel * (UL2Panel - distancePanel) + C2P
anel - pinchSensitivityPanel));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL3Panel && distancePanel < UL3Panel - 5)
  {
    Serial.print(round(A3Panel * (UL3Panel - distancePanel) * (UL3Panel - distancePanel) + B3Panel * (UL3Panel - distancePanel) + C3P
anel - pinchSensitivityPanel));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL4Panel && distancePanel < UL4Panel)
  {
    Serial.print(round(A4Panel * (UL4Panel - distancePanel) * (UL4Panel - distancePanel) + B4Panel * (UL4Panel - distancePanel) + C4P
anel - pinchSensitivityPanel));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL5Panel - 10 && distancePanel < UL5Panel)
  {
    Serial.print(round(A5Panel * (UL5Panel - distancePanel) * (UL5Panel - distancePanel) + B5Panel * (UL5Panel - distancePanel) + C5P
anel - pinchSensitivityPanel));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL6Panel && distancePanel < UL6Panel - 10 )
  {
    Serial.print(round(A6Panel * (UL6Panel - distancePanel) * (UL6Panel - distancePanel) + B6Panel * (UL6Panel - distancePanel) + C6P
anel - pinchSensitivityPanel));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonPanel) == HIGH && distancePanel >= LL7Panel && distancePanel < UL7Panel)
  {
    Serial.print(round(A7Panel * (UL7Panel - distancePanel) * (UL7Panel - distancePanel) + B7Panel * (UL7Panel - distancePanel) + C7P
anel - pinchSensitivityPanel));
    Serial.print("; ");
  }
  else
  {
    Serial.print(0);
    Serial.print("; ");
  }
        // Printing distance, Time_Per_Revolution_Panel and the treshbase
        Serial.print(distancePanel);
        Serial.print("; ");
        Serial.print(Time_Per_Revolution_Panel);
        Serial.print("; ");
        Serial.println(Tresh_base_pos);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the opening of the panel

void DistanceOpenFunkPanel()
{
  progTimePanel = millis();                           // Updated the program time

  if (oneTimeCheckPanel == 0)                         // Enters When there is a hall sensor input
  {
    fullyOpenedTimerPanel = progTimePanel;            // When there is a hall sensor input is the timer reseted
    oneTimeCheckPanel = 1;                            // Sets flag to 1
  }

  // To detect if the panel is at its fully open position
  // Enters if the difference between the program time & the fully open time is less than the specified hard stop time
  if (progTimePanel - fullyOpenedTimerPanel <= hardStopTimePanel)
  {
    if (hsDetectPanel >= hSensitivityPanel)
    {
      distancePanel = distancePanel + confirmerPanel; // Increases the distance
      confirmerPanel = 0;                             // Resets the confirmer
      PaceFunkPanel();                                // Calls the pace function to calculate the time per revolution
      bufferClosePanel = 0;                           // Resets the closing buffer for the hard stop
      fullyClosedPanel = 0;                           // Tells the system that the panel has not reached it fully closed position
      oneTimeCheckPanel = 0;                          // Resets one time check
      bufferOpenPanel = 0;                            // Resets the opening buffer for the hard stop
     }
  }

  // Enters if the difference between the program time & the fully open time is more than the specified hard stop time (to longe betw
een hall sensor input)
```

```
   else if (progTimePanel - fullyOpenedTimerPanel > hardStopTimePanel)
  {
    if (bufferOpenPanel == 0) // Enters if the buffer is 0 --> first time it is to long between Hs inputs
    {
      Serial.println(" --- buffer --- ");             // Prints buffer
      distancePanel = distancePanel + confirmerPanel; // Ingreases the distance
      confirmerPanel = 0;                             // Resets the confirmer
      PaceFunkPanel();                                // Calls the pace function to calculate the time per revolution
      fullyClosedPanel = 0;                           // Tells the system that the panel has not reached it fully closed position
      oneTimeCheckPanel = 0;                          // Resets one time check
      bufferOpenPanel = 1;                            // Sets buffer to 1 so that it doesnt enter tha same statement next time
    }
    else
    {
      Serial.println(" --- PANEL FULLY OPEN --- ");   // Prints That the panel is fully open
      pinchDetectorPanel = 0;                         // Resets the pinch detector to 0 so that the panel can be operated as normal a
gain if a pinch was detected
      distancePanel = fullyOpenDistancePanel;         // Sets the distance to the fully open distance
      progTimePanel = 0;                              // Resets the program time to 0
      fullyOpenedTimerPanel = 0;                      // Resets the fully open timer
      oneTimeCheckPanel = 0;                          // Resets one time check
      fullyOpenPanel = 1;                             // Tells the system that the panel is fully open
      fullyClosedPanel = 0;                           // Tells the system that the panel is NOT fully closed
    }
  }
}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the closing of the panel
void DistanceCloseFunkPanel()
{
  progTimePanel = millis();                           // Updated the program time
  if (oneTimeCheckPanel == 0)                         // Enters When there is a hall sensor input
  {
    fullyClosedTimerPanel = progTimePanel;            // When there is a hall sensor input is the timer reseted
    oneTimeCheckPanel = 1;                            // Sets flag to 1
  }

  // To detect if the panel is at its fully closed position
  // Enters if the difference between the program time & the fully close time is less than the specified hard stop time
   if (progTimePanel - fullyClosedTimerPanel <= hardStopTimePanel)
  {
    if (hsDetectPanel >= hSensitivityPanel)           // Enters when a hall sensor is detected
    {
      distancePanel = distancePanel - confirmerPanel; // Decreases the distance
      confirmerPanel = 0;                             // Resets the confirmer
      PaceFunkPanel();                                // Calls the pace function to calculate the time per revolution
      bufferOpenPanel = 0;                            // Resets the Opening buffer for the hard stop
      fullyOpenPanel = 0;                             // Tells the system that its NOT fully open
      oneTimeCheckPanel = 0;                          // Resets one time check
      bufferClosePanel = 0;                           // Resets the buffer for the closing Hard Stop

      if (distancePanel < stopDistancePanel - 20)     // Enter is if the panel has moved more than 20 steps since the stop in pinch z
one
      {
        stopInPinchZonePanel = 0;                     // Resets stopInPinchZonePanel flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
  }

  // Enters if the difference between the program time & the fully closed time is more than the specified hard stop time (to longe be
tween hall sensor input)
  else if (progTimePanel - fullyClosedTimerPanel > hardStopTimePanel)
  {
    if (bufferClosePanel == 0)                        // Enters if the buffer is 0 --> first time it is to long between Hs inputs
    {
      Serial.println(" --- buffer --- ");             // Prints buffer
      distancePanel = distancePanel - confirmerPanel; // Decreases the distance
      confirmerPanel = 0;                             // Resets the confirmer
      PaceFunkPanel();                                // Calls the pace function to calculate the time per revolution
      fullyOpenPanel = 0;                             // Tells the system that the panel has not reached it fully open position
      oneTimeCheckPanel = 0;                          // Resets one time check
      bufferClosePanel = 1;                           // Sets buffer to 1 so that it doesnt enter tha same statement next time
      if (distancePanel < stopDistancePanel - 20)     // Enter is if the panel has moved more than 20 steps since the stop in pinch z
one
      {
        stopInPinchZonePanel = 0;                     // Resets stopInPinchZonePanel flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
    else
    {
      Serial.println(" --- PANEL FULLY CLOSED --- "); // Prints That the panel is fully closed
      distancePanel = fullyClosedDistancePanel;       // Sets the distance to the fully closed distance
      progTimePanel = 0;                              // Resets the program time to 0
      fullyClosedTimerPanel = 0;                      // Resets the fully closed timer
      oneTimeCheckPanel = 0;                          // Resets one time check
      fullyClosedPanel = 1;                           // Tells the system that the panel is fully closed
      fullyOpenPanel = 0;                             // Tells the system that the panel is NOT fully open
    }
  }
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Identifies input from Hall sensor 1 in Panel
void magnet_detect1Panel()            // The funtions is called by the interrupt when a signal is recived from hall sensor 1
{
  // Enter if any of the buttons are pressed, if there has been a pinch and if the panel is not in its fully closed/open position
  if ((digitalRead(openButtonPanel) == HIGH || digitalRead(closeButtonPanel) == HIGH || pinchDetectorPanel == 1) && fullyOpenPanel ==
 0 && fullyClosedPanel == 0)
  {
    hsDetectPanel++;                  // Steps upp hsDetect
    confirmerPanel++;                 // Steps upp the confirmer
    hs1TimePanel = micros();          // Updates the hall sensor 1 timer
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Identifies input from Hall sensor 2 in Panel
void magnet_detect2Panel()            // The funtions is called by the interrupt when a signal is recived from hall sensor 2
{
 // Enter if any of the buttons are pressed, if there has been a pinch and if the panel is not in its fully closed/open position
  if ((digitalRead(openButtonPanel) == HIGH || digitalRead(closeButtonPanel) == HIGH || pinchDetectorPanel == 1)&& fullyOpenPanel ==
0 && fullyClosedPanel == 0)
  {
    hsDetectPanel++;                  // Steps upp hsDetect
    confirmerPanel++;                 // Steps upp the confirmer
    hs2TimePanel = micros();          // Updates the hall sensor 2 timer
  }
}
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculate the pacing in the Curtain
void paceCurtainFunk()
{
  Time_Between_Hs_Input_Curtain= hs1TimeCurtain - hs2TimeCurtain;           // Inverse of RPM
  Time_Between_Hs_Input_Curtain= abs(Time_Between_Hs_Input_Curtain);        // Take the absolute value

  Time_Per_Revolution_Curtain = Time_Between_Hs_Input_Curtain- Time_Between_Hs_Input_CurtainOld;   // Checks the difference between t
he old and new time difference between hs input
  Time_Per_Revolution_Curtain = abs(Time_Per_Revolution_Curtain);           // Take the absolute value

  hsDetectCurtain = 0;                                                      // Resets Hs detect to 0

 // Digital Filtering
 // If the the time per revolution dropps or increases with more than 1000 from the previous value
 // Is it assumed that it is a false value, and the time per revolution value is set to
 // The old value, same for the distance
  if (Time_Per_Revolution_Curtain  < Time_Per_Revolution_CurtainOld  - 1000 &&  distanceCurtain < UL3Curtain - 10 && distanceCurtain
> LL4Curtain && stopInPinchZoneCurtain == 0 && pinchDetectorCurtain == 0)
  {
    Time_Per_Revolution_Curtain  = Time_Per_Revolution_CurtainOld ;
    distanceCurtain = distanceOldCurtain;
  }
  else if (Time_Per_Revolution_Curtain  > Time_Per_Revolution_CurtainOld  + 1000  &&  distanceCurtain < UL3Curtain - 10 && distanceCu
rtain > LL4Curtain && stopInPinchZoneCurtain == 0 && pinchDetectorCurtain == 0)
  {
    Time_Per_Revolution_Curtain  = Time_Per_Revolution_CurtainOld ;
    distanceCurtain = distanceOldCurtain;
  }

  Time_Between_Hs_Input_CurtainOld = Time_Between_Hs_Input_Curtain;    // Updated the Time_Between_Hs_Input_CurtainOld
  Time_Per_Revolution_CurtainOld  = Time_Per_Revolution_Curtain ;      // Updated the Time_Per_Revolution_CurtainOld
  distanceOldCurtain = distanceCurtain;                                // Updated the distanceOldCurtain

  //Printing distance and Time_Per_Revolution_Curtain
  Serial.print(distanceCurtain);
  Serial.print("; ");
  Serial.println(Time_Per_Revolution_Curtain );
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the opening of the curtain
void DistanceOpenFunkCurtain()
{
  progTimeCurtain = millis();                                   // Updated the program time
  if (oneTimeCheckCurtain == 0)                                 // Enters When there is a hall sensor input
  {
    fullyOpenedTimerCurtain = progTimeCurtain;                  // When there is a hall sensor input is the timer reseted
    oneTimeCheckCurtain = 1;                                    // Sets flag to 1
  }

  // To detect if the curtain is at its fully open position
  // Enters if the difference between the program time & the fully open time is less than the specified hard stop time
   if (progTimeCurtain - fullyOpenedTimerCurtain <= hardStopTimeCurtain)
  {
    if (hsDetectCurtain >= hSensitivityCurtain)                 // Enters when a hall sensor is detected
    {
      distanceCurtain = distanceCurtain + confirmerCurtain;     // Increases the distance
      confirmerCurtain = 0;                                     // Resets the confirmer
      paceCurtainFunk();                                        // Calls the pace function to calculate the time per revolution
      bufferCloseCurtain = 0;                                   // Resets the closing buffer for the hard stop
      fullyClosedCurtain = 0;                                   // Tells the system that the curtain has not reached it fully closed
position
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      bufferOpenCurtain = 0;                                    // Resets the opening buffer for the hard stop
    }
  }

  // Enters if the difference between the program time & the fully open time is more than the specified hard stop time (to longe betw
een hall sensor input)
  else if (progTimeCurtain - fullyOpenedTimerCurtain > hardStopTimeCurtain )
  {
    if (bufferOpenCurtain == 0)                                 // Enters if the buffer is 0 --
> first time it is to long between Hs inputs
    {
      Serial.println(" ---buffer--- ");                         // Prints buffer
      distanceCurtain = distanceCurtain + confirmerCurtain;     // Ingreases the distance
      confirmerCurtain = 0;                                     // Resets the confirmer
      paceCurtainFunk();                                        // Calls the pace function to calculate the time per revolution
      fullyClosedCurtain = 0;                                   // Tells the system that the curtain has not reached it fully closed
position
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      bufferOpenCurtain = 1;                                    // Sets buffer to 1 so that it doesnt enter tha same statement next t
ime
    }
    else
    {

      Serial.println(" --- CURTAIN FULLY OPEN --- ");           // Prints That the curtain is fully open
      pinchDetectorCurtain = 0;                                 // Resets the pinch detector to 0 so that the curtain can be operated
 as normal again if a pinch was detected
      distanceCurtain = fullyOpenDistanceCurtain;               // Sets the distance to the fully open distance
      progTimeCurtain = 0;                                      // Resets the program time to 0
      fullyOpenedTimerCurtain = 0;                              // Resets the fully open timer
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      fullyOpenCurtain = 1;                                     // Tells the system that the curtain is fully open
      fullyClosedCurtain = 0;                                   // Tells the system that the curtain is NOT fully closed
     }
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the closing of the curtain
void DistanceCloseFunkCurtain()
{
  progTimeCurtain = millis();                                   // Updated the program time
  if (oneTimeCheckCurtain == 0)                                 // Enters When there is a hall sensor input
  {
    fullyClosedTimerCurtain = progTimeCurtain;                  // When there is a hall sensor input is the timer reseted
    oneTimeCheckCurtain = 1;                                    // Sets flag to 1
  }

  // To detect if the curtain is at its fully closed position
  // Enters if the difference between the program time & the fully close time is less than the specified hard stop time
 if (progTimeCurtain - fullyClosedTimerCurtain <= hardStopTimeCurtain)
  {
    if (hsDetectCurtain >= hSensitivityCurtain)                 // Enters when a hall sensor is detected
    {
      distanceCurtain = distanceCurtain - confirmerCurtain;     // Decreases the distance
      confirmerCurtain = 0;                                     // Resets the confirmer
      paceCurtainFunk();                                        // Calls the pace function to calculate the time per revolution
      bufferOpenCurtain = 0;                                    // Resets the Opening buffer for the hard stop
      fullyOpenCurtain = 0;                                     // Tells the system that the curtain is NOT fully open
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      bufferCloseCurtain = 0;                                   // Resets the buffer for the closing Hard Stop

      if (distanceCurtain < stopDistanceCurtain - 20)           // Enter is if the curtain has moved more than 20 steps since the sto
p in pinch zone
      {
        stopInPinchZoneCurtain = 0;                             // Resets stopInPinchZoneCurtain flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
  }

 // Enters if the difference between the program time & the fully closed time is more than the specified hard stop time (to longe bet
ween hall sensor input)
 else if (progTimeCurtain - fullyClosedTimerCurtain > hardStopTimeCurtain)
  {
    if (bufferCloseCurtain == 0)                                // Enters if the buffer is 0 --
> first time it is to long between Hs inputs
    {
      Serial.println(" --- buffer --- ");                       // Prints buffer
      distanceCurtain = distanceCurtain - confirmerCurtain;     // Decreases the distance
      confirmerCurtain = 0;                                     // Resets the confirmer
      paceCurtainFunk();                                        // Calls the pace function to calculate the time per revolution
      fullyOpenCurtain = 0;                                     // Tells the system that the curtain has not reached it fully open po
sition
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      bufferCloseCurtain = 1;                                   // Sets buffer to 1 so that it doesnt enter tha same statement next t
ime

      if (distanceCurtain < stopDistanceCurtain - 20)           // Enter is if the curtain has moved more than 20 steps since the sto
p in pinch zone
      {
        stopInPinchZoneCurtain = 0;                             // Resets stopInPinchZoneCurtain flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
    else
    {
      Serial.println(" --- CURTAIN FULLY CLOSED --- ");         // Prints That the curtain is fully closed
      distanceCurtain = fullyClosedDistanceCurtain;             // Sets the distance to the fully closed distance
      progTimeCurtain = 0;                                      // Resets the program time to 0
      fullyClosedTimerCurtain = 0;                              // Resets the fully closed timer
      oneTimeCheckCurtain = 0;                                  // Resets one time check
      fullyClosedCurtain = 1;                                   // Tells the system that the curtain is fully closed
      fullyOpenCurtain = 0;                                     // Tells the system that the curtain is NOT fully open
    }
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funktion that Identifies input from Hall sensor 1 in Curtain
void magnet_detect1Curtain()            // The funtions is called by the interrupt when a signal is recived from hall sensor 1
{
  // Enter if any of the buttons are pressed, if there has been a pinch and if the curtain is not in its fully closed/open position
  if ((digitalRead(openButtonCurtain) == HIGH || digitalRead(closeButtonCurtain) == HIGH || pinchDetectorCurtain == 1) && fullyOpenCu
rtain == 0 && fullyClosedCurtain == 0)
  {
    hsDetectCurtain++;                  // Steps upp hsDetect
    confirmerCurtain++;                 // Steps upp the confirmer
    hs1TimeCurtain = micros();          // Updates the hall sensor 1 timer
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funktion that Identifies input from Hall sensor 2 in Curtain
void magnet_detect2Curtain()            // The funtions is called by the interrupt when a signal is recived from hall sensor 2
{

 // Enter if any of the buttons are pressed, if there has been a pinch and if the curtain is not in its fully closed/open position
  if ((digitalRead(openButtonCurtain) == HIGH || digitalRead(closeButtonCurtain) == HIGH || pinchDetectorCurtain == 1) && fullyOpenCu
rtain == 0 && fullyClosedCurtain == 0)  {

    hsDetectCurtain++;                  // Steps upp hsDetect
    confirmerCurtain++;                 // Steps upp the confirmer
    hs2TimeCurtain= micros();          // Updates the hall sensor 2 timer
  }
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculate the pacing in the Window
void  PaceFunkWindow()
{
  Time_Between_Hs_Input_Window = hs1TimeWindow - hs2TimeWindow;             // Inverse of RPM
  Time_Between_Hs_Input_Window = abs(Time_Between_Hs_Input_Window);         // Take the absolute value

  Time_Per_Revolution_Window = Time_Between_Hs_Input_Window - Time_Between_Hs_Input_WindowOld;// Checks the difference between the ol
d and new time difference between hs input
  Time_Per_Revolution_Window = abs(Time_Per_Revolution_Window);             // Take the absolute value

  hsDetectWindow = 0;                                                       // Resets Hs detect to 0

  Time_Between_Hs_Input_WindowOld = Time_Between_Hs_Input_Window;           // Updated the Time_Between_Hs_Input_WindowOld

// Printing of the treshold curve
if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL1Window && distanceWindow < UL1Window)
  {
    Serial.print(round(A1Window * (UL1Window - distanceWindow) * (UL1Window - distanceWindow) + B1Window * (UL1Window - distanceWindo
w) + C1Window - pinchSensitivityWindow));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL2Window - 5 && distanceWindow < UL2Window)
  {
    Serial.print(round(A2Window * (UL2Window - distanceWindow) * (UL2Window - distanceWindow) + B2Window * (UL2Window - distanceWindo
w) + C2Window - pinchSensitivityWindow));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL3Window && distanceWindow < UL3Window - 5)
  {
    Serial.print(round(A3Window * (UL3Window - distanceWindow) * (UL3Window - distanceWindow) + B3Window * (UL3Window - distanceWindo
w) + C3Window - pinchSensitivityWindow));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL4Window && distanceWindow < UL4Window)
  {
    Serial.print(round(A4Window * (UL4Window - distanceWindow) * (UL4Window - distanceWindow) + B4Window * (UL4Window - distanceWindo
w) + C4Window - pinchSensitivityWindow));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL5Window - 4 && distanceWindow < UL5Window)
  {
    Serial.print(round(A5Window * (UL5Window - distanceWindow) * (UL5Window - distanceWindow) + B5Window * (UL5Window - distanceWindo
w) + C5Window - pinchSensitivityWindow));
    Serial.print("; ");
  }
  else if (digitalRead(closeButtonWindow) == HIGH && distanceWindow >= LL6Window && distanceWindow < UL6Window - 4 )
  {
    Serial.print(round(A6Window * (UL6Window - distanceWindow) * (UL6Window - distanceWindow) + B6Window * (UL6Window - distanceWindo
w) + C6Window - pinchSensitivityWindow));
    Serial.print("; ");
  }
  else
  {
    Serial.print(0);
    Serial.print("; ");
  }
  // Printing distance and Time_Per_Revolution_Window
  Serial.print(distanceWindow);
  Serial.print("; ");
  Serial.println(Time_Per_Revolution_Window );
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Calculated the distance during the opening of the Window
void DistanceOpenFunkWindow()
{
  progTimeWindow = millis();                                   // Updated the program time
  if (oneTimeCheckWindow == 0)                                 // Enters When there is a hall sensor input
  {
    fullyOpenedTimerWindow = progTimeWindow;                  // When there is a hall sensor input is the timer reseted
    oneTimeCheckWindow = 1;                                    // Sets flag to 1
  }
  // To detect if the front window is at its fully open position
  // Enters if the difference between the program time & the fully open time is less than the specified hard stop time
   if (progTimeWindow - fullyOpenedTimerWindow <= hardStopTimeWindow)
  {
    if (hsDetectWindow >= hSensitivityWindow)                 // Enters when a hall sensor is detected
    {
      distanceWindow = distanceWindow + confirmerWindow;     // Increases the distance
      confirmerWindow = 0;                                     // Resets the confirmer
      PaceFunkWindow();                                        // Calls the pace function to calculate the time per revolution
      bufferCloseWindow = 0;                                   // Resets the closing buffer for the hard stop
      fullyClosedWindow = 0;                                   // Tells the system that the front window has not reached it fully closed position
      oneTimeCheckWindow = 0;                                  // Resets one time check
      bufferOpenWindow = 0;                                    // Resets the opening buffer for the hard stop
    }
  }

  // Enters if the difference between the program time & the fully open time is more than the specified hard stop time (to longe between hall sensor input)
  else if (progTimeWindow - fullyOpenedTimerWindow > hardStopTimeWindow)
  {
    if (bufferOpenWindow == 0)                                 // Enters if the buffer is 0 --> first time it is to long between Hs inputs
    {
      Serial.println(" ---buffer--- ");                        // Prints buffer
      distanceWindow = distanceWindow + confirmerWindow;       // Ingreases the distance
      confirmerWindow = 0;                                     // Resets the confirmer
      PaceFunkWindow();                                        // Calls the pace function to calculate the time per revolution
      fullyClosedWindow = 0;                                   // Tells the system that the front window has not reached it fully closed position
      oneTimeCheckWindow = 0;                                  // Resets one time check
      bufferOpenWindow = 1;                                    // Sets buffer to 1 so that it doesnt enter tha same statement next time
      }
    else
    {
      Serial.println(" --- WINDOW FULLY OPEN --- ");           // Prints That the front Window is fully open
      pinchDetectorWindow = 0;                                 // Resets the pinch detector to 0 so that the window can be operated as normal again if a pinch was detected
      distanceWindow = fullyOpenDistanceWindow;                // Sets the distance to the fully open distance
      progTimeWindow = 0;                                      // Resets the program time to 0
      fullyOpenedTimerWindow = 0;                              // Resets the fully open timer
      oneTimeCheckWindow = 0;                                  // Resets one time check
      fullyClosedWindow = 0;                                   // Tells the system that the window is NOT fully closed
      fullyOpenWindow = 1;                                     // Tells the system that the window is fully open
    }
  }
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Calculated the distance during the closing of the Window
void DistanceCloseFunkWindow()
{
  progTimeWindow = millis();                                   // Updated the program time
  if (oneTimeCheckWindow == 0)                                 // Enters When there is a hall sensor input
  {
    fullyClosedTimerWindow = progTimeWindow;                   // When there is a hall sensor input is the timer reseted
    oneTimeCheckWindow = 1;                                    // Sets flag to 1
  }

  // To detect if the front window is at its fully closed position
  // Enters if the difference between the program time & the fully close time is less than the specified hard stop time
  if (progTimeWindow - fullyClosedTimerWindow <= hardStopTimeWindow)
  {
    if (hsDetectWindow >= hSensitivityWindow)                  // Enters when a hall sensor is detected
    {
      distanceWindow = distanceWindow - confirmerWindow;       // Decreases the distance
      confirmerWindow = 0;                                     // Resets the confirmer
      PaceFunkWindow();                                        // Calls the pace function to calculate the time per revolution
      bufferOpenWindow = 0;                                    // Resets the Opening buffer for the hard stop
      fullyOpenWindow = 0;                                     // Tells the system that the window is NOT fully open
      oneTimeCheckWindow = 0;                                  // Resets one time check
      bufferCloseWindow = 0;                                   // Resets the buffer for the closing Hard Stop

      if (distanceWindow < stopDistanceWindow - 20)            // Enter is if the window has moved more than 20 steps since the stop in pinch zone
      {
        stopInPinchZoneWindow = 0;                             // Resets stopInPinchZoneWindow flag to 0 so that anti-pinch works as normally after the stop
      }
    }
  }
  // Enters if the difference between the program time & the fully closed time is more than the specified hard stop time (to longe between hall sensor input)
 else if (progTimeWindow - fullyClosedTimerWindow > hardStopTimeWindow)
  {
    if (bufferCloseWindow == 0)                                // Enters if the buffer is 0 --> first time it is to long between Hs inputs
    {
      Serial.println(" --- buffer --- ");                      // Prints buffer
      distanceWindow = distanceWindow - confirmerWindow;       // Decreases the distance
      confirmerWindow = 0;                                     // Resets the confirmer
      PaceFunkWindow();                                        // Calls the pace function to calculate the time per revolution
      fullyOpenWindow = 0;                                     // Tells the system that the window has not reached it fully open position
      oneTimeCheckWindow = 0;                                  // Resets one time check
      bufferCloseWindow = 1;                                   // Sets buffer to 1 so that it doesnt enter tha same statement next time
      if (distanceWindow < stopDistanceWindow - 20)            // Enter is if the window has moved more than 20 steps since the stop 
in pinch zone
      {
        stopInPinchZoneWindow = 0;                             // Resets stopInPinchZoneWindow flag to 0 so that anti-
pinch works as normally after the stop
      }
    }
    else
    {
      Serial.println(" --- WINDOW FULLY CLOSED --- ");          // Prints That the window is fully closed
      distanceWindow = fullyClosedDistanceWindow;             // Sets the distance to the fully closed distance
      progTimeWindow = 0;                                      // Resets the program time to 0
      fullyClosedTimerWindow = 0;                              // Resets the fully closed timer
      oneTimeCheckWindow = 0;                                  // Resets one time check
      fullyOpenWindow = 0;                                     // Tells the system that the window is NOT fully open
      fullyClosedWindow = 1;                                   // Tells the system that the window is fully closed
    }
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funktion that Identifies input from Hall sensor 1 in the front window
void magnet_detect1Window()            // The funtions is called by the interrupt when a signal is recived from hall sensor 1
{
   // Enter if any of the buttons are pressed, if there has been a pinch and if the window is not in its fully closed/open position
  if (digitalRead(openButtonWindow) == HIGH || digitalRead(closeButtonWindow) == HIGH || pinchDetectorWindow == 1)
  {
    hsDetectWindow++;                  // Steps upp hsDetect
    confirmerWindow++;                 // Steps upp the confirmer
    hs1TimeWindow = micros();          // Updates the hall sensor 1 timer
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Appendix H

Circuit diagram for concept 1 is shown in Figure 62.

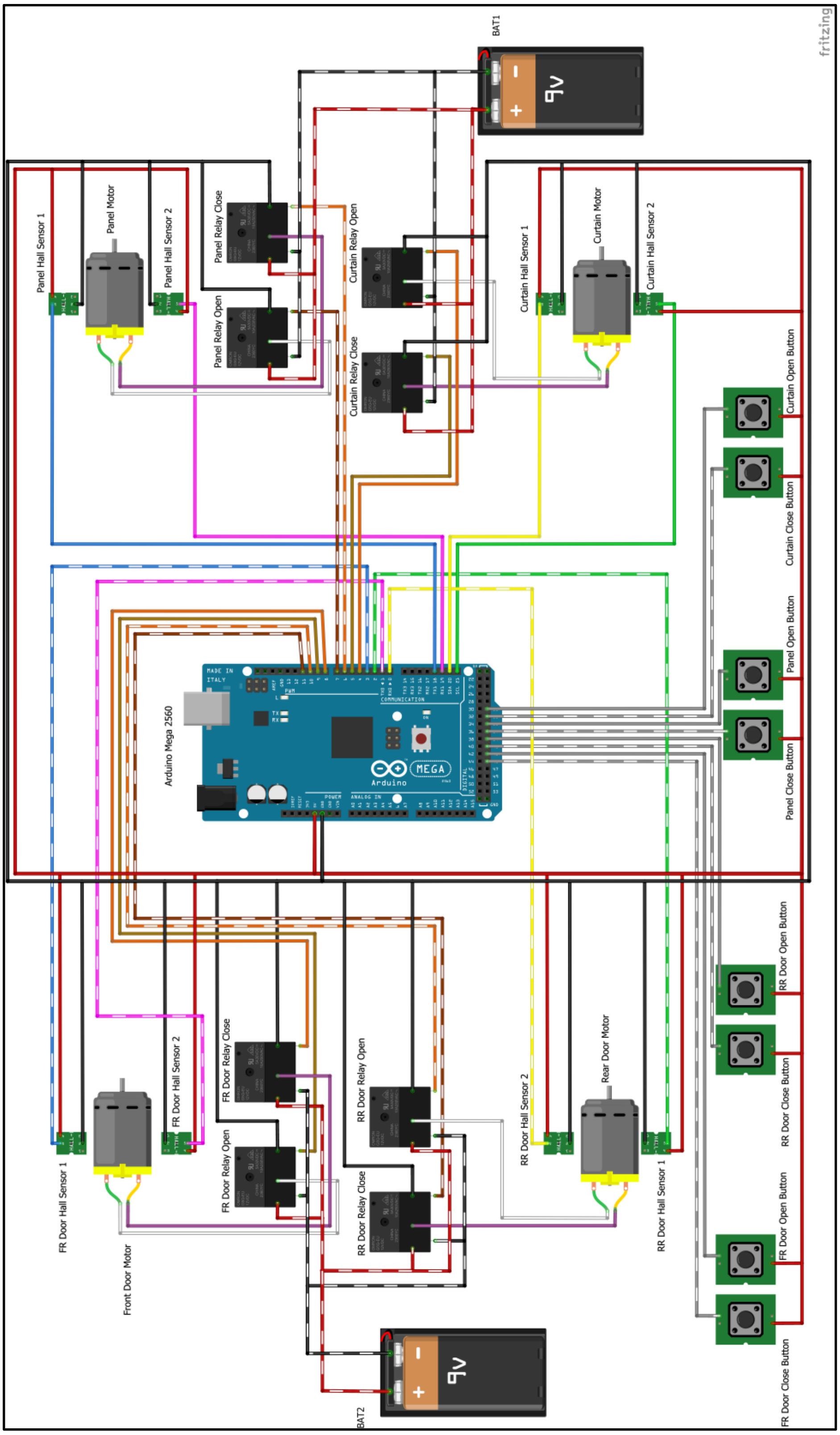


Figure 62 Complete circuit diagram for concept 1 for sunroof panel, curtain, front and rear windows

Circuit diagram concept 2 is shown in Figure 63.

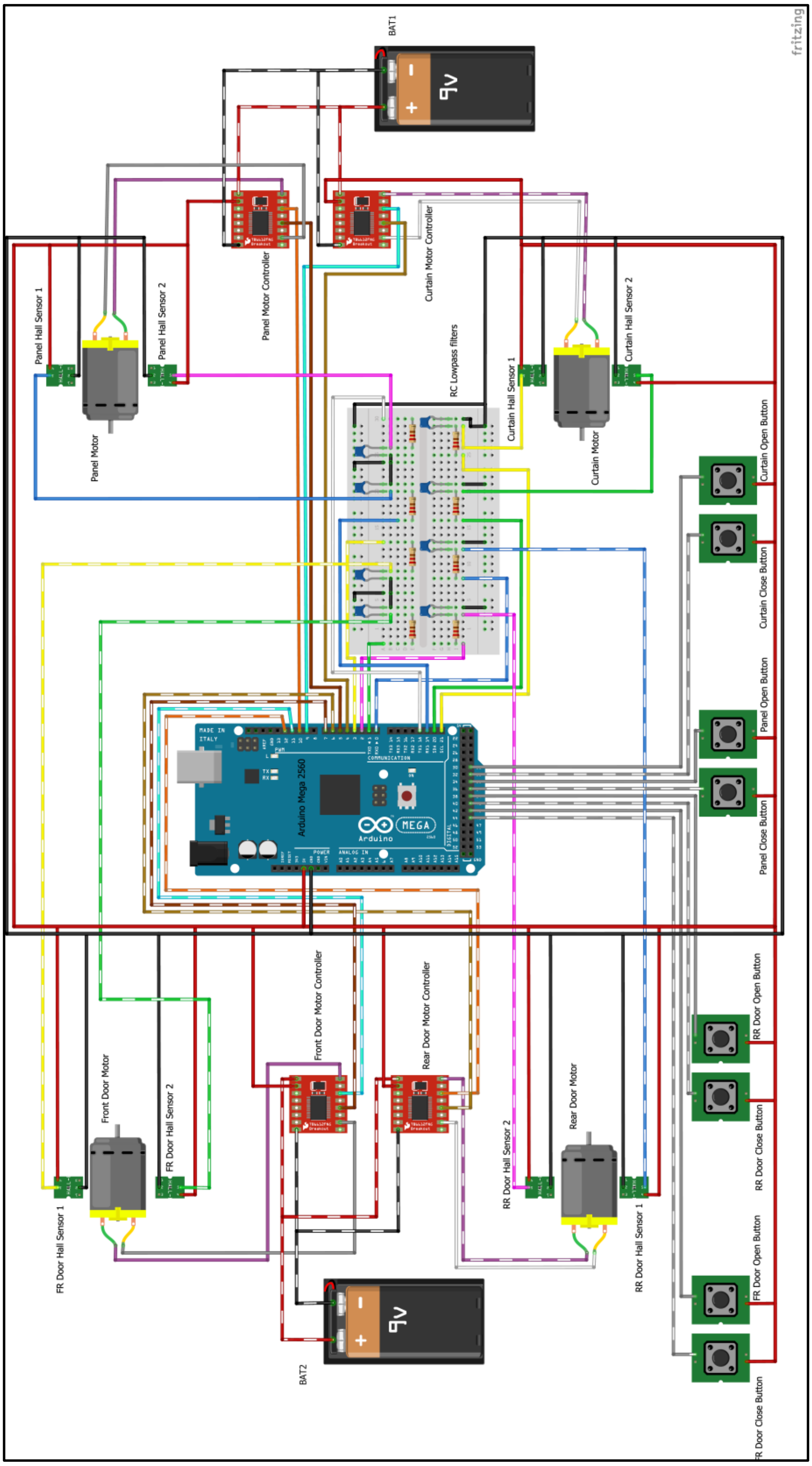


Figure 63 Complete circuit diagram for concept 2 for sunroof panel, curtain, front and rear windows

DEPARTMENT OF INDUSTRIAL AND MATERIAL SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY