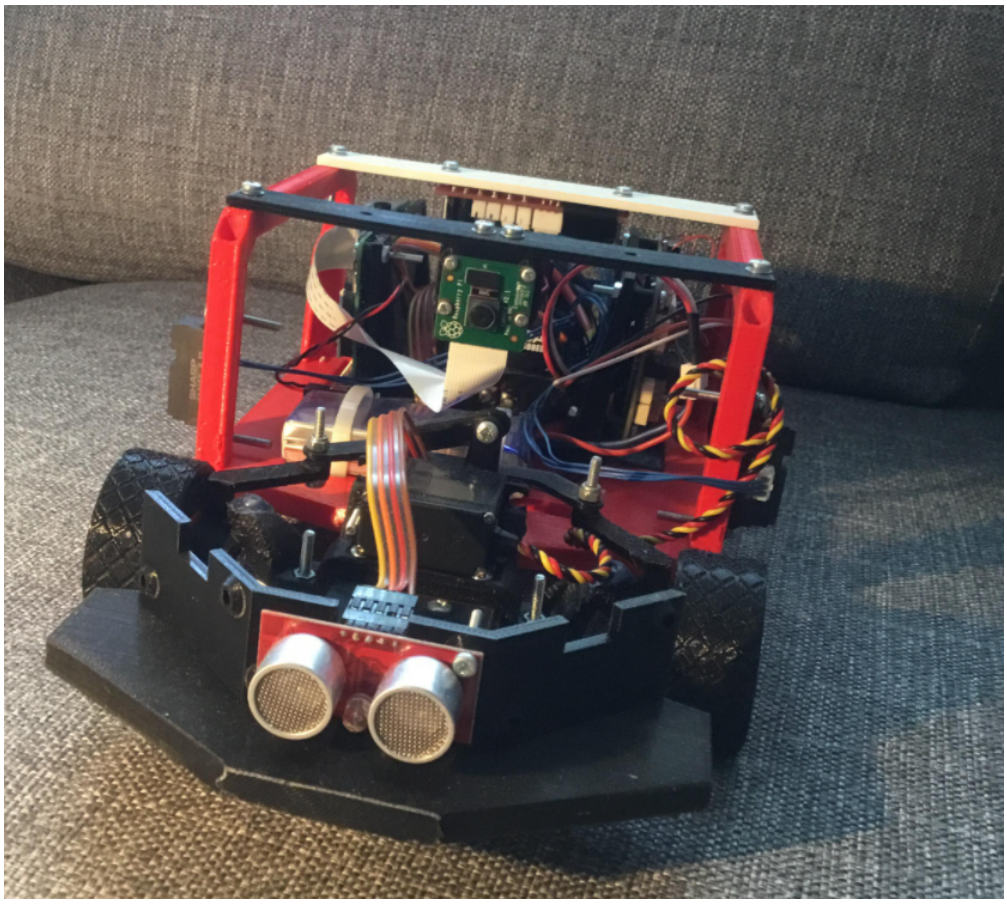


Stimulate learning for autonomous vehicles

From building the Kiwi car to implementation of software

Master's thesis in Master Programme Complex adaptive systems

Anton Jansson

Department of Mechanics and maritime sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Stimulate learning for autonomous vehicles

Anton Jansson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and maritime sciences
Division of vehicle engineering and autonomous systems
Chalmers University of Technology
Göteborg, Sweden 2021

Stimulate learning for autonomous vehicles
From building the Kiwi car to implementation of software
Anton Jansson

Master's Thesis 2021:74
Department of Mechanics and maritime sciences
Vehicle engineering and autonomous systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 31 772 1000

Cover: The Kiwi car

Typeset in LaTeX
Printed by Chalmers Digitaltryck
Göteborg, Sweden 2021

Stimulate learning for autonomous vehicles
From building the Kiwi car to implementation of software
Anton Jansson
Division of vehicle engineering and autonomous systems
Chalmers University of Technology

Abstract

In this thesis, the so called Kiwi platform was studied as a tool for teaching autonomous systems and OpenDLV to four target groups: secondary education, undergraduate education, postgraduate education, and hobbyists. The Kiwi platform consists of a 3D printed car equipped with sensors for measuring distances as well as a camera for image processing. Moreover, the open source software environment OpenDLV, was used to develop and test algorithms for driving the Kiwi car autonomously. The project was divided into two main parts, starting with building the Kiwi car and then programming it. Afterwards a survey was conducted in order to evaluate this teaching concept. Only one test group was involved in building the car, while all groups programmed it. In order to build the car, the primary resource was step by step instructions. When programming the car, the test groups were given a selection of mandatory topics. The steps were presented both in text as well as with video tutorials. Based on the results from the survey the participants found the tutorials relevant and extensive enough for the four programming tasks. Having said that, only the two test groups with the expected highest prior knowledge completed all the tasks in time, which may indicate that the tutorials may have been to advanced. Furthermore, the negative feedback from the survey were regarding the more technically challenging sections. The test groups with highest expected prior knowledge found that the material as a simplification of their intent, this was not expressed by the other test groups. Regarding the user manual for building the Kiwi car, the data was not extensive enough to draw any general conclusions.

Keywords: programming, Kiwi car, autonomous driving, teaching, online community

Acknowledgements

First, I would like to thank my supervisor Ola Benderius for discussing ideas on how the project should be designed to reach the long-term goal of developing an online community for autonomous vehicles. Secondly, I wish to thank Björnborg Nguyen for providing parts and ordering missing parts for the Kiwi car as well as for providing the cars with the latest software updates. A thanks also goes to Krister Blanch, Sara Larsson, Lage Bonde and Niklas Liljegren, the main persons behind the test groups programming and building the Kiwi car.

Anton Jansson, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

During the last decade, the leading car companies have experienced a transition where novel activities and processes focusing on driving assistance has become crucial. Traditionally, the companies have been focused on developing stronger and more effective engines, but today's technical equipment, such as collision avoidance and self-driving systems, are the most important selling points for the manufacturers. The importance of technical solutions can also be seen when nontraditional companies such as Apple and Google invest in developing their own cars. The fact that pure technology companies see an opportunity to enter an already competitive market indicates how critical the technical solutions will be for surviving in the vehicle industry.

Autonomous vehicles will probably revolutionize the way the society are used to transporting goods or people. This will likely result in both social and economic benefits from minimizing car accidents, reducing travel time as well as more efficient parking and fuel consumption. Combined, these improvements are estimated to approach between US$ 2,000–4,000 per *autonomous vehicles* (AV) and year [1]. Furthermore, traffic accidents are the ninth leading cause of fatality world-wide, resulting in over 1.2 million people losing their lives and leaving between 20 to 50 million people to suffer non-fatal injuries. Unfortunately, these numbers will probably increase. By 2030, traffic accidents are estimated to be the fifth world leading cause of death, causing 2.3% of total deaths. Developing safe autonomous solutions that can greatly reduce the effects of human error can therefore save millions of lives each year [2]. However, approaching this direction in vehicle development raises other concerns. Shifting the responsibility for the decision making from humans to computers is already disputed from an ethical point of view. Can the system be made secure enough to keep it from being compromised, and can we trust it to never fail? There are many problems to solve before these kinds of systems can be reliable enough to replace human supervision. This is just two of many issues which must be addressed before computers can replace humans.

How autonomous and safe our vehicles will be in the future is in the hands of skilled problem solvers and software developers. Nevertheless, for most people, programming is something very abstract and tricky. One way to make it less abstract could be to create an online community to spark interest in programming among young people as well as challenge students with new concepts within the field. However, creating an online community, targeting people with different levels of prior knowl-

edge and different ages, is challenging. To obtain effective learning in online communities, self-regulation and motivation among the students are critical factors. If self-regulation and motivation are further divided into subgroups, continuous guiding and support turns out to be key factors for success [3]. Having said that, factors that are not related to the material itself, but rather to personal circumstances may also effect the learning outcome. Students from privileged areas tend, for instance, to succeed in online education to a higher degree than their counterparts [4].

At Chalmers University of Technology, efforts are being made to encourage young students interest in the field by developing a platform for autonomous vehicles. The platform consists of an open-hardware 3D-printable miniature car, called Kiwi, and the open-source microservice-based software framework OpenDLV. The car is equipped with IR and ultrasonic sensors covering the four sides of the car, and one camera facing forwards. The main purpose of OpenDLV is to test and develop solutions for self-driven vehicles [5].To increase flexibility, all microservices are built with Docker-hub, which allows the developer to run the programs independent of the operating system. For the communication between the microservices, libcluon is used as the middleware. Libcluon is the world's first and only single-file, header-only middleware for distributed systems and robotic applications [6].

1.1 Purpose

Programming with the Kiwi platform, libcluon, OpenDLV and Docker-hub makes it possible to work in a similar environment as modern vehicle companies with continuous integration, over the air updates, and feedback loops on software performance. This is beneficial when it comes to measuring product performance and distributing software updates.

This project aims to increase the knowledge in, and stimulate learning about autonomous vehicles to a wider group of people. The main target groups are secondary education, undergraduate education, postgraduate education, and hobbyists. The project is divided into two parts. The first part is related to constructing the Kiwi car from scratch, and the second part focuses on tutorials for programming it. The programming tutorials are based on theory, such as continuous integration and microservice based architecture, combined with coding examples and corresponding exercises. In the last exercise of the tutorials the Kiwi car should be able to drive autonomously. Using image detection to recognize cones by the roadside, the car should be able to adjust its yaw angle and speed if necessary.

The project aims to answer the following research questions:

1. How can the Kiwi platform be used to teach OpenDLV to a heterogeneous online community, including students from secondary education, undergraduate education, postgraduate education, and hobbyists?
2. What are the minimum requirements on the web content for such teaching and learning, and how can this content be made available to all four target groups in a user-friendly way?

3. What coding examples and specific tutorial steps are most efficient for teaching with the Kiwi platform, while also teaching OpenDLV and microservice-based autonomous vehicle software in general?

1.2 Limitations

The project is focused on providing instructions and tutorials for building and programming the Kiwi car. Furthermore, the project should be seen as the first step in exploring ideas and learning materials for building an online community for autonomous vehicles. Creating a web page where the material will be distributed or improve the already existing Kiwi car is therefore not in the scope of the project. The instructions for building the Kiwi car should be self-explanatory except for the components that have to be purchased separately. For this project, the required components were provided by Chalmers for the test groups. The programming tutorials cover a selection of topics that are considered necessary for working with the Kiwi car. The sections in the tutorial are:

- Setting up the environment
- Booting the system
- Connecting to Wifi
- Controlling the robot
- Writing the first code
- The first programming task
- Pushing and compiling code on Gitlab
- Downloading the image to the Kiwi car
- A second programming task
- Image detection
- Data replay
- Cone detection
- Finding the position of the cones
- Task 3
- Task 4

The postgraduate education (PhD) and undergraduate education (master) test groups were given three and a half weeks to complete the tutorials, while the hobbyist group was given approximately two months. The secondary school group was given two and half months to both build and program the Kiwi car.

2
Background

This section provides a background for autonomous vehicles and strategies for effective learning, where the second topic is divided into two subgroups. The first part introduces general strategies for learning, and the second one focuses on how these strategies can be implemented in online learning.

2.1 Autonomous vehicles

Autonomous vehicles have been illustrated as a science fiction scenario in movies, but the recent progress towards driver-less cars, indicates that they might be driving on our streets relatively near in the future. The increasing competition between traditional car companies (such as Volkswagen, Toyota, Volvo) and information technology companies (such as Apple and Google) has accelerated the the technological development, but the idea of driver-less cars is nothing new. On the contrary, ever since vehicles were first invented futurists have been dreaming about computerized driving. One of the first demonstrations took place in the 1920s. The car was then controlled by a radio signal, similar to a radio car, but this required a second car to send the signal [7].

A few decades later, ideas about embedded electronic devices in the roadway were born. The devices would activate the driver-less cars when the condition was beneficial. The idea was considered in the UK and parts of the US, but re-designing the roads to include the electronic devices was too expensive [7].

In the 1960s the focus was again redirected to the cars. Around 20 years later, Ernst Dickmanns and his team managed to drive a Mercedes Benz autonomously over 20 km with a top speed of 96 km/h on an empty highway. Then, in the late 1980s cars from Mercedes Benz were able to recognise some obstacles and in the 1990s it could change lanes autonomously [7]. The technological progress sparked many new projects and events, both within academia and the industry. One of the events was for example the *defense advanced research projects administration* (DARPA) which held a competition where the vehicles should drive 380 km autonomously within ten hours. Unfortunately, the competition was no success the first year and the vehicles could only drive a few kilometers before crashing [8].

Today's systems are more reliable but there are technical problems to be solved before implementing autonomous vehicles in a large scale. One of them is the map-

ping system, which must have a significant better accuracy than the GPS guidance or other mapping systems that are frequently used today. In 2015, Google had mapped around 2,000 miles and their test car had shown good results on these roads [7]. However, the reactions of these algorithms outside the mapped roads was not tested. Difficulty for autonomous vehicles is that they need to be able to drive in all weather conditions. Driving in snow, heavy rain, on ice or in troubling light conditions are still challenges that autonomous vehicles are facing. Nevertheless, the biggest problem is probably to ensure that no unauthorized actors can break through the security system of the vehicles [7].

2.2 Strategies for learning

Our ability to learn and apply knowledge has given us an advantage compared to other species, including the technology that makes our lives more comfortable. However, modern technology also has its drawbacks. For example it has lead to new challenges that need to be solved to avoid extinction of the majority of life on our planet. To solve these problems education plays an important role. Unfortunately, there is no universal way of learning that suits everyone. On the contrary, well functioning and well-adapted learning styles are highly individual.

In the early and mid-20th century behaviorism dominated the educational landscape [9]. Knowledge was seen as small discrete blocks that the student used to build her understanding, and the more blocks, the more knowledge is possessed. During the mid and later part of the 20th century, theories within constructivism increased in popularity. The theories supported cognitive thinking, exploring and problem solving rather than to learn by repetition. In the early state the individual learning process was dominating, giving rise to social constructivism, which questioned whether learning was an individual process [10].

Today, active learning and feedback are recommended as key components for effective learning [11]. John Hatties report, based on 500 meta-analyses, involving 450,000 effect sizes[1] from 180,000 studies, representing approximately 20 to 30 million students, conclude that the main purpose of feedback is to reduce discrepancies between current understandings and performance and goal. To reduce the gap, the following three questions should be asked: where am I going? (feed up), how am I going? (feedback) and where to go next? (feed forward). To answer the first question Hattie emphasizes that a critical aspect is to give the student information about their attainment of learning goals related to the task. The type of goal, if it is about passing a test, performing a task or to do better than last time is not the essential. Instead, challenge and commitment are of greater importance. The student will then be able to evaluate their actions and efforts accordingly.

In the second question “how am I going?” a mentor is most likely involved and

[1] A score calculated by Hattie that helps prioritize which variables that affect learning the most [12].

the type of feedback used is critical for the student's learning. Given the average effect of schooling (0.40), Hattie shows that the most effective methods for feedback were cues (1.10) or reinforcement to learners (0.94), while programmed instruction (-0.04) and praise (0.14) were less effective or could even have a negative effect. The average effect of feedback was 0.79, almost twice the average effect of schooling.

In the third question "where to go next?", Hattie is critical to the idea of giving more information, more tasks and expectations. Instead, he encourages more self-regulation over the learning process, greater fluency and automaticity, and more strategies and processes to work with the tasks to get a deeper realization of what is understood and what is not [13].

Active learning and student-centered approaches have been proven to be effective learning strategies [11]. The use of the 5E model (engagement, exploration, explanation, elaboration, evaluation) is one example of how learning can become more active. In the engagement phase, the teacher or a curriculum task accesses students' prior knowledge or curiosity to help them become engaged in the new concept of the lecture. This could for example be achieved by smaller activities or explicit prior knowledge. In the exploration phase the students are activated with tasks where misconceptions, processes and skills are identified. The phase that is most comparable with traditional teaching is the explanation phase. The supervisor can for example introduce new concepts or processes, but a common thread through the five steps is active participation by the students. In the elaboration phase the students get to practice their understanding of the concept by conducting exercises or additional activities. Lastly the students' progress is evaluated. The evaluation form can for example be a project presentation, lab report or a quiz [14].

One technique to increase the activity based learning is to integrate a method called *think-share-pair* (TSP) into the lectures. In an undergraduate science class at the university of Colorado-Boulder, the students used clickers to evaluate their understanding of the topic taught. The students were told to answer the questions individually (Q_1) before discussing with their neighbors. After the discussion they were told to answer the same question again (Q_{ad}). Next, the students were instructed to answer an isomorphic question individually (Q_2). Almost all student that answered Q_1 correctly, also did so in Q_2. Out of the students who answered Q_1 incorrectly and Q_{ad} correctly, 77% answered Q_2 correctly. For the students answering incorrectly in both Q_1 and Q_{ad}, 44% still managed to answer Q_2 correctly. For all the click questions, the average percentage of correct answer for Q_2 was significantly higher than for Q_1 and $Q1_{ad}$ [15].

2.2.1 Learning online

With the ongoing Corona pandemic students have been forced to adapt to new ways of learning. Instead of going to the school or university, learning is to greater extent taking place online. In a study, based on research for e-learning and education

in international high-impact scientific journals between 2009 and 2018, the most frequently used keyword was *massive open online course* (MOOC). The research could be divided into three subgroups where one of them were e-learning and online students, which could be divided further into two main themes: (1) self-regulation, and (2) dropout and retention [16].

Motivation and self-regulation turned out to be striking key factors for success. By using *educational data mining* (EDM) to analyze 412 students' access behavior in e-learning it was shown that only 10% of the group could be considered as active learners [3]. Self-regulation on the other hand is considered as a skill that students need to improve by guidance [17] [18]. How well they succeed also depends on their background. By studying online education's effect on students of different gender, social background and previous education, it could be concluded that social background and previous education are important factors when it comes to the students' performance. The study concludes three success factors namely: if the student was living in a privileged area, their previous education, and how active the students were during the lectures. Nevertheless, the study did not find any connections between gender and success [4].

How active the students are, is essential to their degree of success. Analyses of 17,934 server logs from 98 undergraduate students show that students who were less likely of accessing course materials, writing or reading messages and taking part in discussions also received a lower final grade. On the other hand, the group of students that were most active did not perform the best among the student groups [19]. This shows the complexity of online education and that the common thread for success is self-regulation and motivation. One important step for maintaining or increasing motivation is continuous guiding and support. On the other hand, poor quality of e-learning materials, technology incompetence and technical problems are just a few examples of what may have the opposite effect [3].

3

Theory

This section gives an introduction to the Kiwi platform and the software environment OpenDLV, which has as its main purpose to test and develop algorithms for self-driving vehicles. The explanation of the Kiwi platform and OpenDLV are of basic nature and only the most frequently used sensors of the Kiwi car and the most vital concepts of OpenDLV will therefore be covered in this section.

3.1 Kiwi platform

The Kiwi platform consists of two parts: the Kiwi car itself (which is shown in Fig 3.1) and the teaching assistance behind it. The Kiwi car is equipped with sensors and a camera. Four distance sensors are located on the car, consisting of two IR sensors, one on either side of the car, and two ultrasonic sensors located at the front and the rear of the car. Additionally, a Raspberry Pi camera is located at the roof to record data that can be used to detect objects ahead.

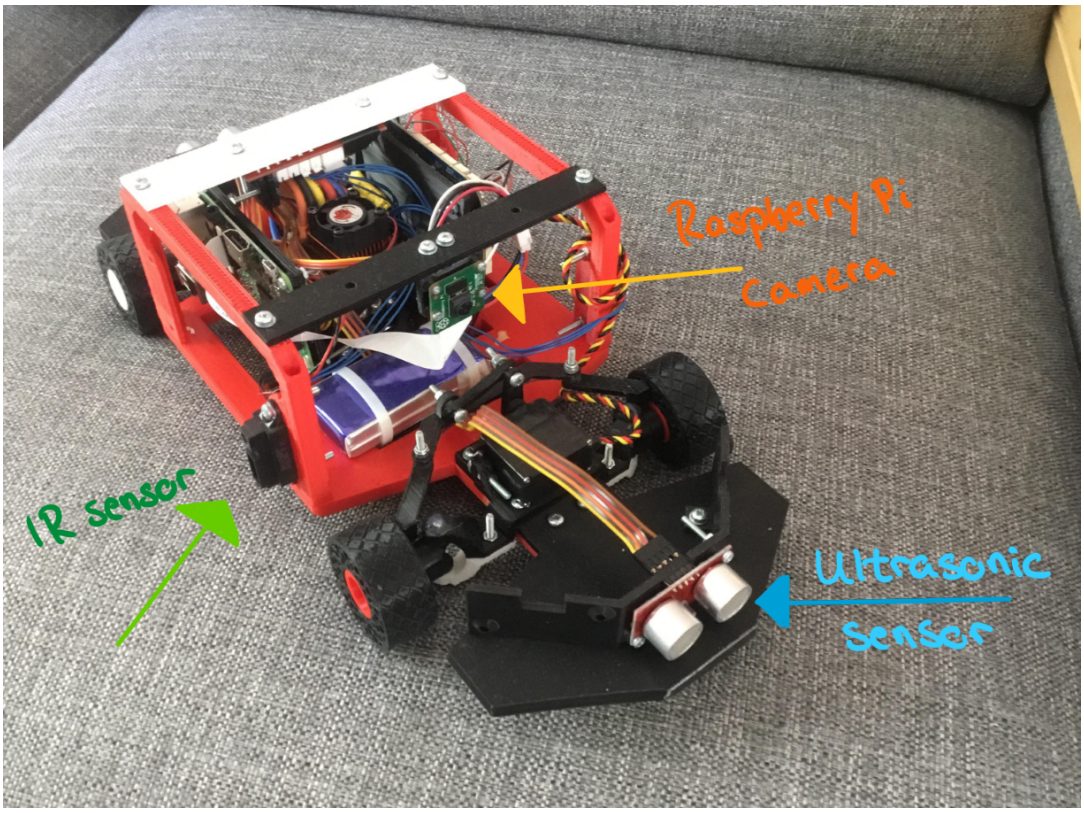


Figure 3.1: Illustration of the Kiwi car. Four distance sensors are located on the car, consisting of two IR sensors, one on either side of the car, and two ultrasonic sensors located at the front and the rear of the car. Additionally, a Raspberry Pi camera is located in the front.

3.2 OpenDLV

OpenDLV is an open-source software environment in which the main purpose is to support the development and testing of self-driving vehicles. The environment is entirely based on microservices, which can be described as well-defined single task applications that are isolated from other microservices running on the server. Since the microservices are single task oriented with a well-defined input and output, the update frequency of new features can be increased. The reason is that only the microservice in question needs to be updated and pushed. Since they are isolated, they will not interfere with other microservices running on the server [20] [21].

Another important tool is Docker, which creates a container that consists of the application itself including a small operative system. Since the operating system is included, the application will work in the exact same way in any environment, therefore increases its deployability.

Lastly, OpenDLV relies on libcluon for communication between the microservices. Libcluon is a single-file, header-only that is independent of external libraries which can easily be included into the project [6].

4
Methods

This chapter presents how the data from the survey was collected and analysed. The thesis consisted of two parts, building the Kiwi car and creating tutorials for programming it. Each part is presented separately below. The manual can be found in Appendix D, the tutorials in Appendix E and the corresponding survey questions in Appendix A.1 and A.2 respectively.

4.1 Building the Kiwi car

For building the car a secondary school class was chosen as a test group. Since the instructions were designed to be comprehensible by all test groups, the test group expected to have the lowest prior knowledge was chosen. The secondary school class was divided into different groups were each group was given the task of assembling different parts of the car. The work was divided amongst smaller groups to avoid the situation where a few students would end up doing all the work, which may have resulted in feedback only representing a small high performing part of the class. By having smaller groups, discussions may take place where students with different learning patterns can identify lack of information or parts that need to be explained in more detail. After completing each section in the manual, the students answered a survey to evaluate to which extent the manual was helping them assembling the parts.

4.1.1 Survey questions

The survey consisted of two mandatory questions and one voluntary. Both the mandatory and voluntary questions can be found in Appendix A.1. The mandatory questions were given a scale from one to five, where one means that you do not agree with the statement, and five that you fully agree. The focus of the mandatory questions was to investigate whether the manual was understandable and if the information given was enough to fulfil the task. The questions were repeatedly the same for each section of the manual to increase the reliability when identifying which sections that were easy to understand and which needs to be further improved. Since the survey was taken after each section, the mandatory questions were created with the intention of being minimized in both number, complexity and length of answer. The reason for this is to make sure that the feedback corresponds to which degree the manual is transferring the knowledge, rather than how many times you have answered the questions. In other words, having to many questions in each

section may degrade the quality of the answers. Scaled questions were preferable in comparison to other answer alternatives. Using scaled questions was estimated to result in the best comprehension between how the test group experienced the different tasks, since the respondents data is given by a distinct number.

A voluntary question was used as a feedback channel for the test group to come with ideas for improvement from their point of view, but also highlight which sections were most difficult to understand. The answers were not meant to evaluate if the manual was successful in transferring the knowledge or not. Instead the question aims to identify further improvements, which explain why the answers are given in text format rather than numbers.

4.1.2 How to analyze the data and its quality

The data was analyzed with focus on answering the first research question, how the Kiwi platform can be used to teach OpenDLV to a heterogeneous online community. Since only one of the four target groups was using the manual and only one group within the selected target group was given the opportunity to build the Kiwi car, no comparison between groups could be made. The result may also have been affected by which of the persons in the test group it was that performed the task. The data was meant to be analyzed in a way to identify the less pedagogical parts and further improvements of the manual, but also how well it managed to transfer the knowledge in each section. This was supposed to be done by comparing the feedback between the different performed tasks, but also by identifying the experienced difficulties from the voluntary questions. Since only feedback from the first steps was received the analysis is of fundamental nature and will not reflect the overall impression of the manual.

4.2 Programming the Kiwi car

Four persons, one from each target group specified in the research questions, programmed the car following the tutorials. Each group borrowed a car from Chalmers to exclude that mistakes from building the Kiwi car could propagate into programming it. After performing one or several steps, depending on their extent, the test groups were asked to participate in a survey to evaluate how well the tutorials guided them through the tasks. Some questions have been constructed as statements for the test groups to evaluate. All the questions can be found in Appendix A.2.

4.2.1 Survey questions

There are three questions that are repeatedly the same for all the sections. The questions are:

- How advanced did you experience the material?
- How relevant was the material from your interests?
- Suggestions on improvements

The first two questions are based on a scale from one to five where one means very easy/not relevant and five means very difficult/very relevant for the first and second question, respectively. In the third question the student can write suggestions in text format. All the questions are mandatory. These questions target all three research questions and aim to measure how well the material managed to find a balance between challenging the test groups with more experience and guiding the groups that were not as familiar with programming, but also to identify what can be improved. From the same perspective the questions also examined how relevant the material was from the test groups interests.

The first question was analyzed to identify the knowledge gap between the test groups. This was done by comparing the overall score from the survey from each test group and compare them to each other. The questions were also analyzed individually where all the test groups' answers were summed up with the intention of identifying sections that were more challenging. If the survey outcome for a specific question was low, it may indicate that the section was not pedagogical enough, too easy or too difficult. To specify what part of the material that can be improved the text questions were analyzed. This was done with the purpose of finding a pattern, both in and between the test groups, to find points of future improvement in the material.

The one to five scale, explained above, was implemented in both the first and second question. The second question was investigating how relevant the material was compared to the test groups' experience and their interests. The result was used to evaluate if the tutorials cover the necessary steps with respect to the test groups interests and if not, identify what is missing.

The survey questions related to a specific research question are presented individually below.

4.2.2 Questions targeting the first research question

The focus of the first research question is how the Kiwi platform can be used to teach OpenDLV in an online community to different target groups. The survey questions are mainly focusing on three areas. First, the essential steps to get started with the Kiwi platform such as installing Ubuntu, booting the Kiwi car and connecting to WiFi. Secondly, how effective external links and material are to challenge the target groups that have a better prior knowledge. Thirdly, how well prepared the students are for performing the exercises.

For the essential steps a mandatory question was used where the student may decide whether the text, links and tutorials were useful in guiding them to fulfill the steps in the tutorial. The questions are graded from one to five, where one means that the survey participant totally disagree and five means that the survey participant totally agrees. The statements are adapted to the content of the section but are similar to:

- The links and tutorials guide the user to download the necessary programs

The questions scaled from one to five, were used to identify the steps that were more challenging. To get a more specific explanation of what can be improved the same text question as presented in section 4.2.1 was used. It was desirable that the students explained in text format what information was missing to complete the task or give feedback on improvements.

To be able to challenge the student with better prior knowledge, when it comes to working with the Kiwi platform, links to external sources, where more details could be found about the topic, were used. The survey was used to conclude how effective the links were to stimulate further learning. The statement could for example be:

- Links to further explore docker on your own is helpful. If you did not use the external links, skip this question.

The question is scaled from one to five as described above and the student may decide if the links were useful for further exploration in the topic. No further voluntary question was used where the student could specify if they used the external links or not. Instead, these specific questions that are directed towards a specific test group was set as voluntary. In each survey there is at least one text question where the student can share their general thoughts about the material and further improvements.

To sum up the information presented during the tutorials, exercises with a corresponding questionnaire was used. The questions were not focusing on to what degree the student managed to solve the task. The area of focus was to understand whether they, from the task description, could understand what they were supposed to do and if it was in line with the presented information in the earlier steps. The statements were:

- The information presented in the previous section is enough to understand what to do in the task
- The task is in the line with the information presented in the previous sections

The questions were not designed to be influenced by how the student choose to solve the task, but instead focus on if the student had the tools to solve it. The survey also includes free text questions where the student can express if they experience that some steps were missing.

4.2.2.1 Analyze the result for the first research question

The data obtained from the essential steps and the exercises, were analyzed with the intention to find the lowest level of knowledge within the test groups. When constructing an online community, keeping students motivated and giving them guidance and support is a key factor for success. Nevertheless, simplifying the content too much may instead give the opposite effect. The mandatory questions

were designed to indicate how the four different test groups experienced the content while the voluntary question highlighted which parts that were most difficult to understand.

Regarding the questions related to external links and how to get a deeper understanding of the topics, not all test groups are expected to answer these questions. Since the same questions are appearing several times where different types of websites are used, the data will be analyzed in order to identify what kind of external sources are most suitable for an online community.

4.2.3 Questions targeting the second research question

The second research question focuses on the minimum requirements for web content and how it can be presented in a user-friendly way.

The survey questions were therefore focusing on two areas: if the steps presented covered the minimum requirements for being able to program the Kiwi car, and if the information was presented in a user-friendly way. For the question related to the layout of the material the same scaled questions were used as in section 4.2.2. The students were asked if the layout of mixing text, text boxes and lists are giving the student a good overview as well as helping them to identify the most important steps. The statement was:

- The layout of mixing text, text boxes, lists and tutorial give a good overview and helps the reader identify the most important steps.

To avoid having many questions in each survey, sections where the layout is estimated to have a greater impact on the learning rate, were chosen.

To evaluate if the material covered the minimum requirements the survey questions related to the exercises were used. The questions are the same as described in section 4.2.2 and focus on the overall expression, not only the layout, but also the content and how it is presented in text. The question aims to identify if the students experience that the tutorials contain the minimum requirements to understand the task and if they have the tools to solve it.

4.2.3.1 Analyze the result for the second research question

The sections where the test groups were asked about how they experience the layout are quite similar in design. All sections are using some text together with text boxes and lists. This is a consequence of the low number of students taking the survey. Instead of analyzing different approaches to how the content can be designed in a user-friendly way, this study was focusing on how the amount of text in relation to text boxes and lists were affecting the student's ability to learn the material. This was conducted by both comparing the outcome on the surveys but also analyzing if sections where text is more frequently used are more likely to cause errors for the

test groups and if one can conclude that this is a consequence of the layout or of the difficulties of the subject.

To analyze if the tutorials contain the minimum requirements for being able to programming the Kiwi car, the same method as described in section 4.2.2.1 was used. The questions related to the exercises were compared between the test groups to identify if the tutorials contained the minimum requirements or which parts that need to be improved. Similarly, the recurrent questions about how advanced the test groups experienced the material was used to estimate the balance in the tutorials. In other words, if some sections were considered generally too easy or too difficult, the text questions were used to identify key areas that were considered more challenging.

4.2.4 Questions targeting the third research question

The third research question is focusing on what coding examples and specific steps that would be the most efficient to teach students about the Kiwi platform and OpenDLV. The questions follow the same standard as described in section 4.2.2 and the key statements were:

- The information presented in the previous section is enough to understand what to do in the task
- The task is in line with the information presented in the previous sections

For the tasks related to image detection, hints were presented. These parts had two extra statements.

- The hints were useful
- The hints exposed to many problems that the student could figure out on their own

The questions have the same scaled structure as described before, where one means totally disagree and five totally agree.

The questions related to the coding example are focusing on how helpful the tutorials were when the students developed their code, and if the balance between hints and instructions was good. When students are taking online courses, the balance between guidance and support but also the student's ability to self-regulate is critical for the learning outcome. The first question, if the coding example was helpful when developing the code, is targeting how guiding and supportive the tutorials are while the question related to the balance between hints and instructions aims to answer if the students were challenged enough.

To evaluate which steps were the most efficient when teaching the Kiwi platform and OpenDLV the survey questions were divided into two branches. The first branch focused on if the video tutorials and instructions were helpful to understand the topics and if the motivation of, for example Docker and the purpose of the three

Docker commands, was clear. These questions aim to give an indication whether the instructions and video tutorials transfer the content of the lecture in question. Together with the questions related to the exercises, the questions aim to give a better picture of which specific tutorials are beneficial for the learning outcome.

The second part of the survey questions were directed to the sections *sources of error* and if they were useful to solve or identify what may cause problems. The statements were voluntary and was formulated as follow:

- If you used the "source of error". It was useful to solve or identify what may have caused the problem

These questions aim to answer to which degree the identified errors help the user to solve or identify the problem and then search for a solution online. Covering all possible errors that may occur is not possible, instead these questions should be seen as guidelines for how to prioritize future work.

4.2.4.1 Analyze the result for the third research question

When analyzing the questions related to the coding example, the main focus were on the balance between instructions and the student's ability to solve the task on their own. The feedback from the test groups were compared in between them to identify if the coding example would need to be explained further, for example with coding tutorials for similar problems.

To analyze which steps in the tutorials that would be most efficient, both the response from the questions related to the exercise and how useful the instructions and video tutorials were for guiding the student was taken into account. The result was used to identify sections that were experienced as more challenging. The questions related to the *sources of error* were used as an indication to estimate if further work should focus on identifying more of the possible errors that the students may face or rather focusing on expanding the material with more content and exercises.

5

Results

This chapter presents the results of the thesis, where the evaluation for building the Kiwi car and programming the car are presented separately. The test group building the Kiwi car was unable to finish it due to unexpected events, and therefore only the first steps of the manual are presented. Three of the test groups started programming the Kiwi car and two of them completed all the steps during the timeline for the project. In the figures (5.1–5.10) presented below, *section* refers to the sections in the manual or tutorials. The sections are presented in chronological order, but the name may be shortened to increase the readability. In appendix C, the complete section names for the programming tutorial can be found for each abbreviation. To minimize the labels in the figures, the two test groups "postgraduate education" and "undergraduate education" will be referred to as PhD and master respectively.

5.1 Building the Kiwi car

The first question of the questionnaire is *The manual provided enough information to perform the specified steps.* In figure 5.1 the survey question outcome is shown, where the score "1" means "Totally disagree" and the score "5" means "Totally agree". Note especially that the test group (secondary school) only completed the two initial steps, and therefore the result should not be seen as a representation of the general result for the manual. Instead, more data needs to be collected to evaluate how well the manual guides the user in assembling the Kiwi car.

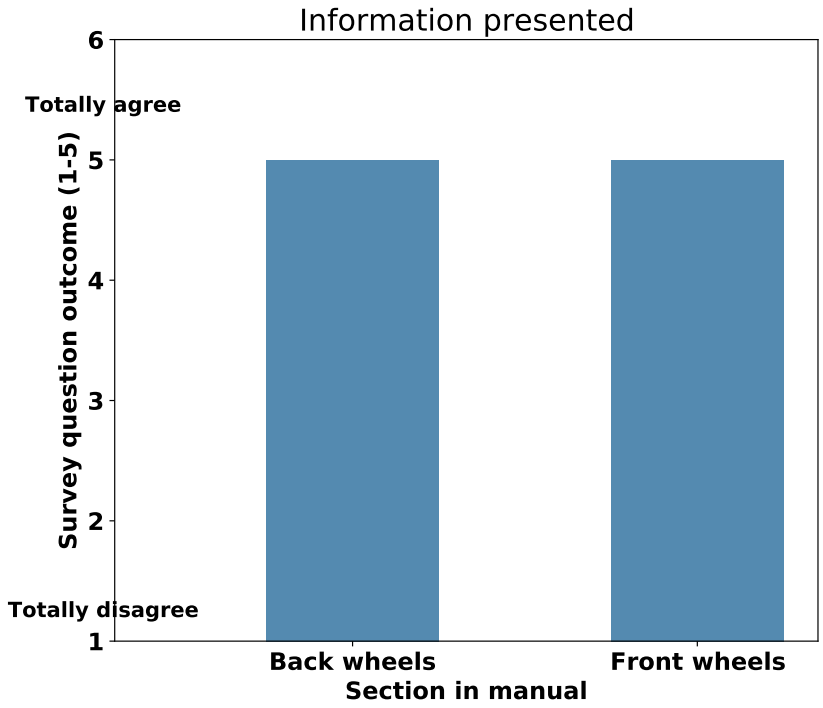


Figure 5.1: A diagram showing the results for the question *The manual provided enough information to perform the specified steps.* Note that only the two initial steps were completed.

The second question of the questionnaire is *It was possible to understand what was to be done.* Fig. 5.2 displays the survey question outcome, where the score "1" means "Totally disagree" and the score "5" means "Totally agree". Note that the results are only valid for the front and back wheels and should not be seen as the general result of the manual.

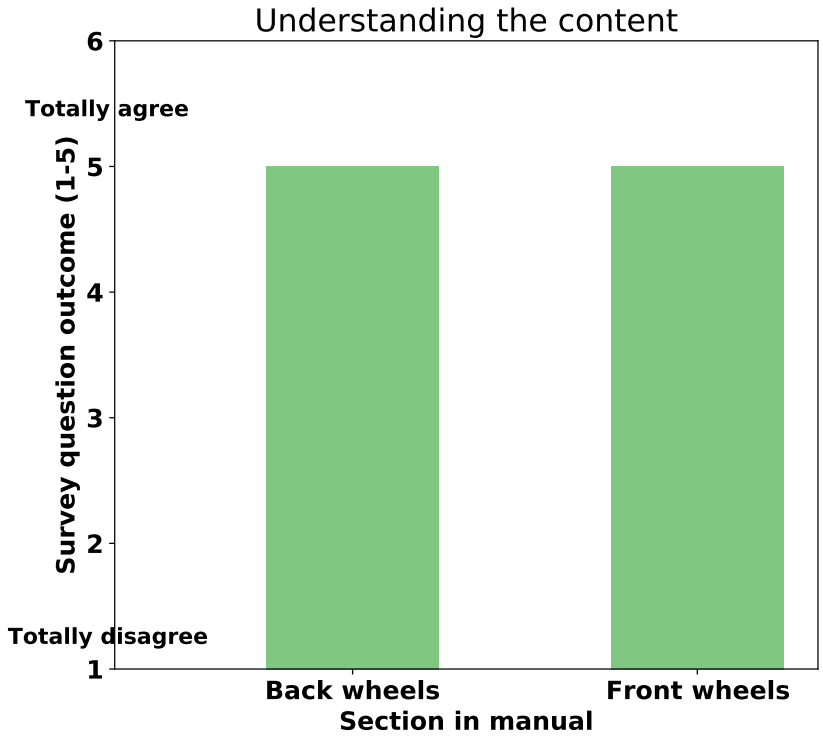


Figure 5.2: A diagram showing the results for the question *It was possible to understand what was to be done.* Note that the result only are valid for the front and back wheels and should not be seen as the general result of the manual.

5.2 Programming the Kiwi car

The survey outcome from the question *The information presented in the previous section is enough to understand what to do in the task* is illustrated in Fig. 5.3. The x-axis corresponds to the tasks in the tutorials and y-axis shows the survey outcome received by the respondents. The scale for the survey is one to five, where the score "1" means "Totally disagree" and the score "5" means "Totally agree". As illustrated in the figure, most of the test groups agreed with the statement. Note the low score for task one (see Fig. 5.3) in the master test group.

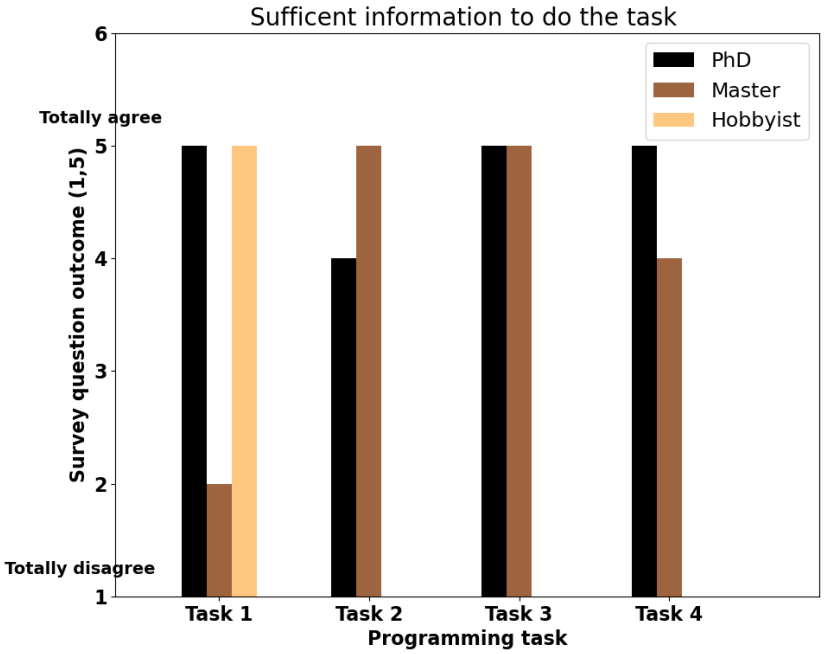


Figure 5.3: A diagram showing the result of how well prepared the students were for the tasks. The question the respondents were answering was: *The information presented in the previous section is enough to understand what to do in the task.* Most of the test groups agreed with the statement, except for the master test group in task one.

The survey outcome from the questions *The hints were useful* and *The hints exposed to many problems that the student could figure out on their own* is shown in Fig. 5.4. The label "messages", shown on the x-axis, refers to the code examples related to how one sets up the communication channel, as well as sending and receiving messages from the Kiwi car. The second label "detect", also presented at the x-axis, refers to the code where one uses image detection to detect the cones, their position and how to steer the Kiwi car. The scale for the survey is one to five, where the score "1" means "Totally disagree" and the score "5" means "Totally agree". It can be seen that the balance between hints and the student's ability to figure out the more challenging parts on their own differ between the test groups. In the more challenging parts (at the end of the tutorial), one can observe how the test groups with lower expected prior knowledge were more positive to the provided hints.

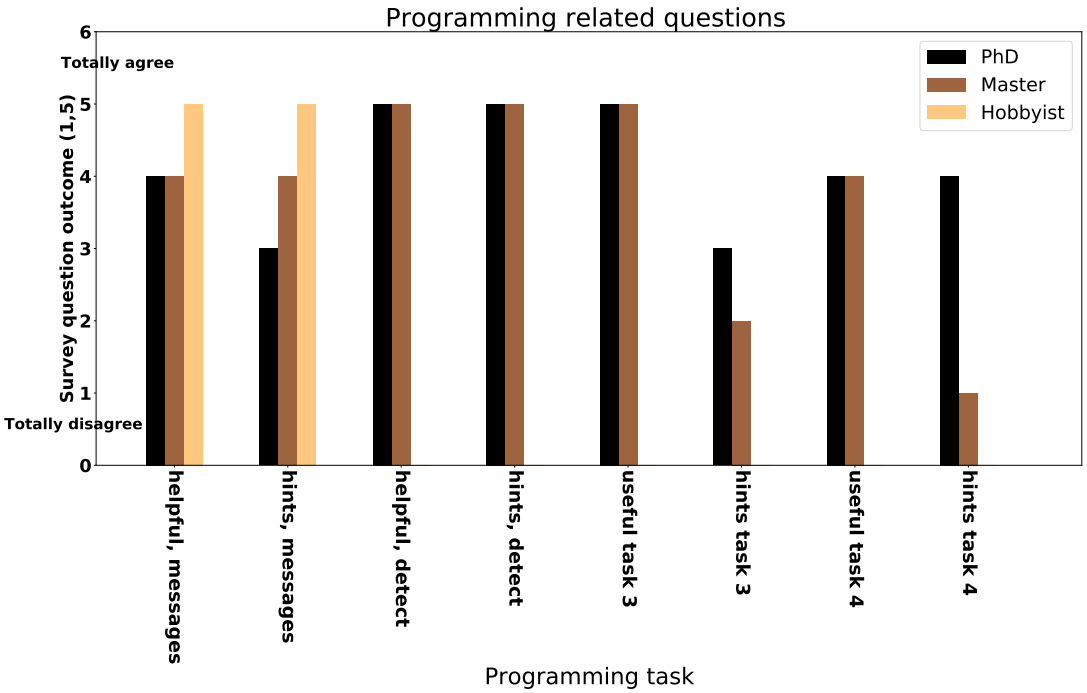


Figure 5.4: A diagram showing the survey outcome from the questions *The hints were useful* and *The hints exposed to many problems that the student could figure out on their own*. The test groups with lower expected prior knowledge were more positive to the hints given.

The survey outcome from the question *How advanced did you experience the material?* is presented in Fig. 5.5. The x-axis shows the sections in the tutorial and the y-axis presents the survey outcome given by the respondents. The scale for the survey is one to five, where score "1" means "Very easy" and score "5" means "Very hard". The grey curve represents the mean value for each section. Note that the mean value is based only on two or three data points. It can be seen that the degree of difficulty increases through the tutorials. After the second task, the test groups could choose between a more challenging path or a simpler one, which is why the mean value is illustrated by a dashed line. The PhD test group chose the more challenging path while the master test group chose the simpler one, which may explain the observed difference in perceived difficulty in the later steps. Note that the line for the hobbyist test group stops after the section Push code. The reason is that the hobbyists did not have enough time to complete all tasks under the given time frame. Fig. 5.6 shows the mean value with corresponding standard deviation for the respective test group, based on the data in Fig. 5.5. The scale is therefore following the same standard as in Fig. 5.5. The number N refers to the number of survey questions answered by each test group.

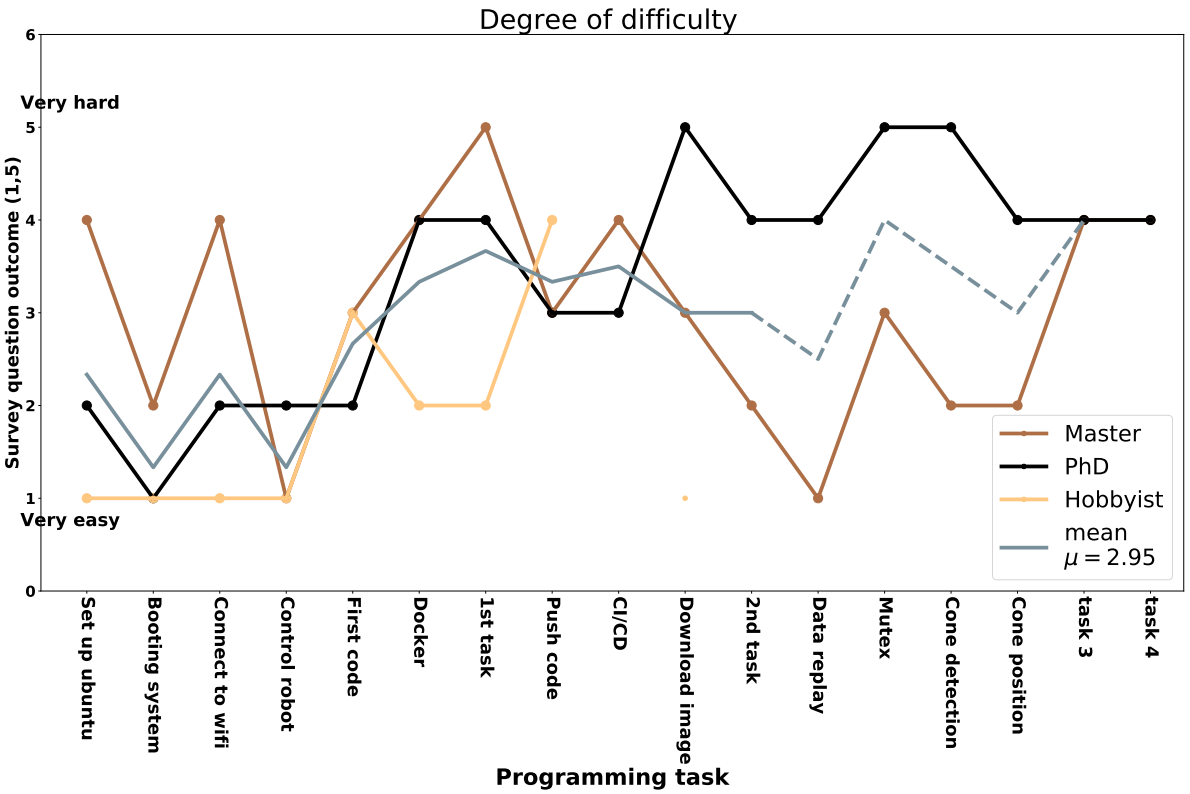


Figure 5.5: Perceived level of difficulty of the material for the different test groups with respect to each section of the tutorial. The question the respondents were answering was: *How advanced did you experience the material?*. After the second task, each group were given the option to pursue a more challenging route. The mean value is therefore indicated by a dashed line in the figure. Furthermore, the hobbyist test group did not completed all the steps during the time frame, which is why the line stops after Push code.

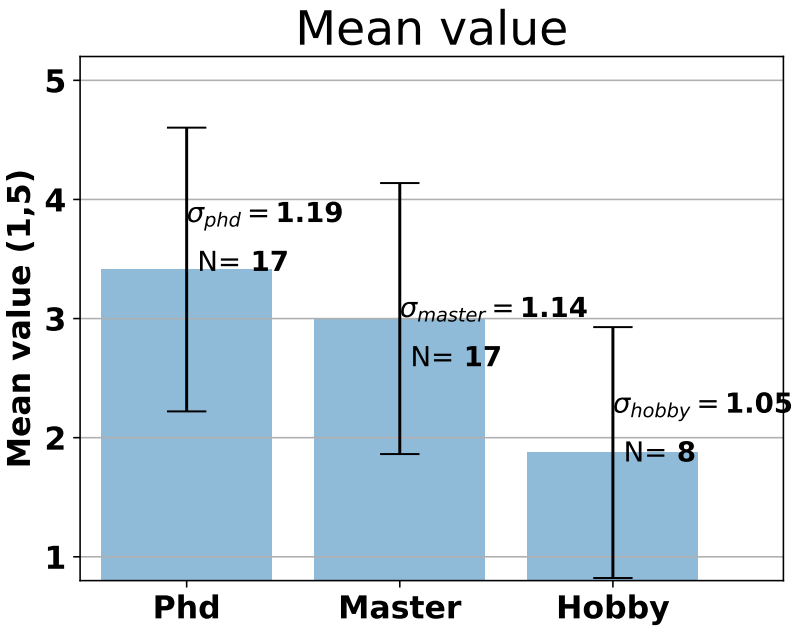


Figure 5.6: Mean value with corresponding standard deviation for the survey outcome in Fig. 5.5 for respective test group. The number N refers to the number of survey questions answered by each test group.

The survey outcome from the question *How relevant was the task from your interests* is displayed in Fig. 5.5. The x-axis shows the sections in the tutorial and the y-axis presents the survey outcome given by the respondents. The scale for the survey is one to five, where "1" means "Not relevant" and "5" means "Very relevant". The blue line illustrates the mean value. Note that the mean value is based on two or three data points. One can see that most of the sections were considered relevant to all the test groups, with an overall mean value of 4.81 out of 5. The hobbyist test group found the most steps less relevant than the other test groups, especially the steps in the beginning of the tutorial. Both the PhD and master test groups completed the tutorial within the time frame, while the hobbyist test group completed half of the tasks. In Fig. 5.8 the mean value with corresponding standard deviation can be seen for the respective test groups, based on the data in Fig. 5.7. The number N refers to the number of survey questions answered by each test group.

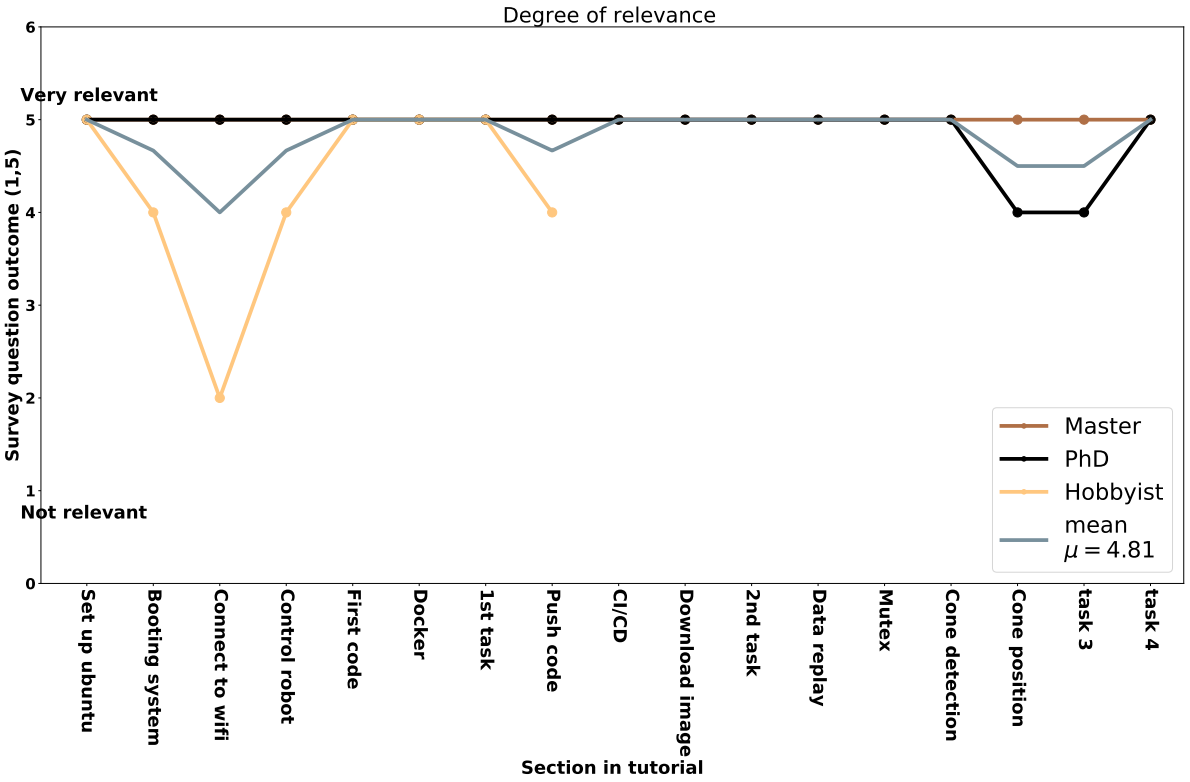


Figure 5.7: Perceived level of relevance of the material for the different test groups with respect to each section of the tutorial. The question the respondents were answering was: *How relevant was the task from your interests.* The overall mean value is high ($\mu = 4.81$ out of 5) with only one outcome which received a score lower than four. Furthermore, the hobbyist test group did not completed all the steps during the time frame, which is why the line stops after Push code.

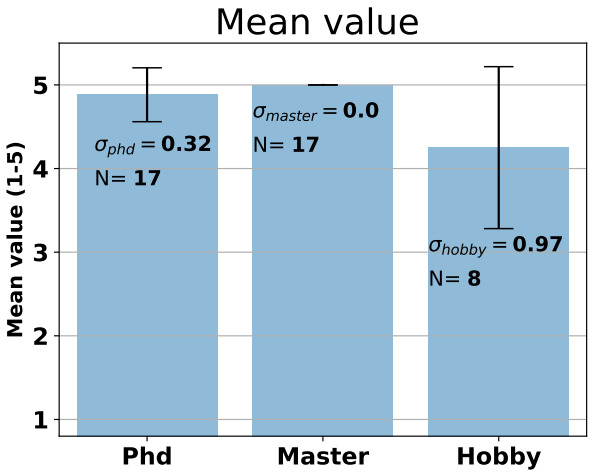


Figure 5.8: Mean value with corresponding standard deviation for the survey outcome in Fig. 5.7 for respective test group. The number N refers to the number of survey questions answered by each test group. The mean value is close to five, at least for the two test groups completed all the steps in the tutorial.

The survey outcome from the question *If the layout of mixing text, text boxes, lists and tutorial gives a good overview and helps the reader identify the most important steps* is shown in Fig. 5.9. As illustrated in the x-axis, the question was asked four times. In these sections the amount of text, text boxes and lists differ in between. The y-axis presents the survey outcome given by the respondents, where the score "1" means "Totally disagree" and the score "5" means "Totally agree". It was observed that the mean value does not differ to much between the sections, but the PhD test group is in general more skeptical to the layout than the master test group. An interesting aspect is that the Docker section received the lowest score from the PhD test group, while the highest score were received from the master and hobbyist test groups. Note that the hobbyist test group did not complete the tutorials in the time frame, which is why line stops after the section Push code.

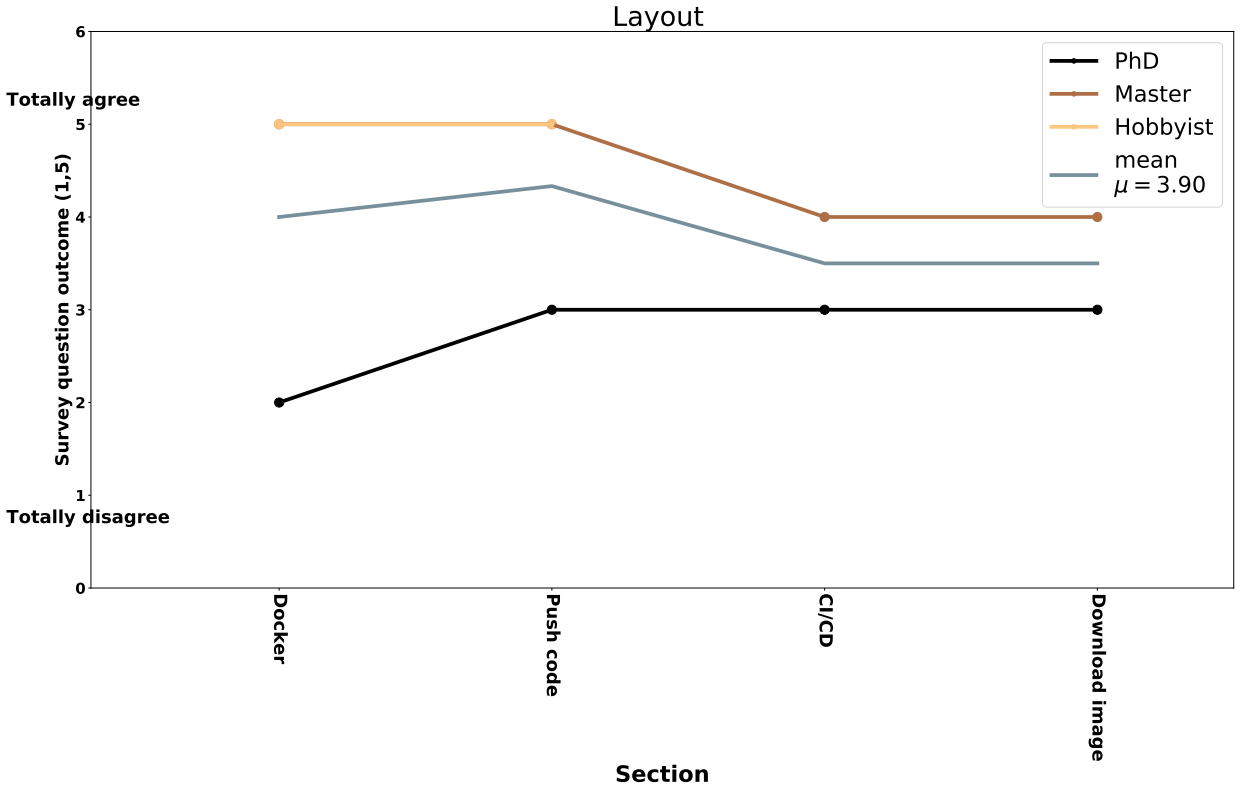


Figure 5.9: Illustration of to which extent the layout gives the user a good overview and helps the reader to identify the most important steps in the sections. The question the respondents were answering was: *If the layout of mixing text, text boxes, lists and tutorial gives a good overview and helps the reader identify the most important steps.* Four sections, where the amount of text, text boxes and lists differ in between was chosen. Note that the PhD test group was in general more skeptical to the layout than the master and hobbyist test group. The section that the PhD test group was most critical to obtained the highest outcome from both the master and hobbyist test group. Note that the hobbyist test group did not complete the tutorials during the time for this project, which explains why the line stops after the section Push code.

The survey outcome from the questions similar to *The instructions and tutorial is guiding the user do connect to the Wifi* is presented in Fig. 5.10. Note that the questions are adapted to the context and therefore differ between the sections presented at the x-axis. The y-axis presents the survey outcome given by the respondents, where the score "1" means "Totally disagree" and the score "5" means "Totally agree". The blue line illustrates the mean value. Note that the mean value is based on two or three data points. The overall mean value is high ($\sigma = 4.42$ out of 5.0), but sections as Docker and CI/CD received a low value from the PhD test group. It can be observed that the lower expected prior knowledge the test group had, the higher survey outcome was received. Note that the hobbyist test group did not completed the tutorial during the time for this project, which is why the line stops

after section Push code.

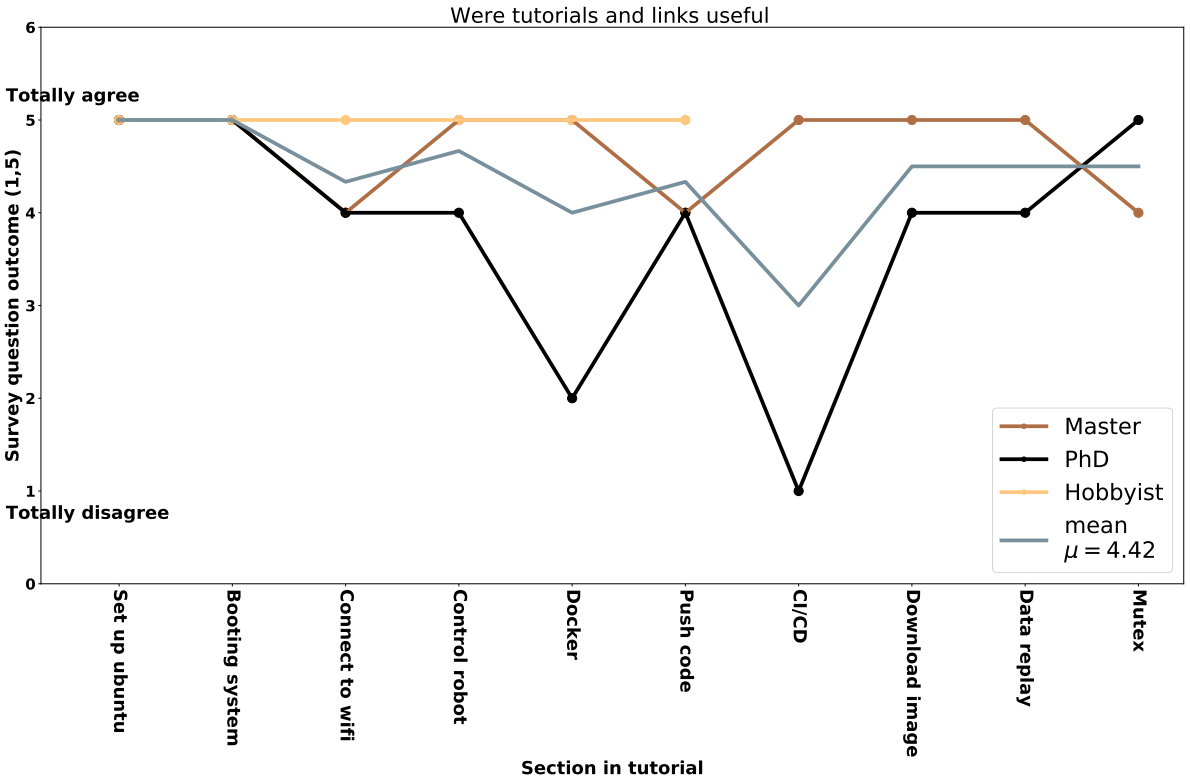


Figure 5.10: Perceived level of guidance of the links and tutorials for the different test groups with respect to each section of the tutorial. The question the respondents were answering was similar to: *The instructions and tutorial is guiding the user do connect to the Wifi*, but was adapted to the context of the section. It was observed that the mean value is high ($\sigma = 4.42$ out of 5.0), but two sections were given low survey outcome from the PhD test group. Generally, the test groups with the expected lowest prior knowledge was more positive to the links and tutorials. Note that the hobbyist test group did not complete the tutorial during the time for this project, which is why the line stops after section Push code.

Tab 5.1 presents a selection of text answers that were of importance when analysing the results in Fig. 5.3– 5.10 and identifying future improvements in the tutorial. The text questions the respondents were answering were *Suggestions on improvements* and *If the coding example was not enough, what were you missing?* The first text question was present in every survey, while the second only was asked when coding examples were demonstrated. For instance, the first row in the table is regarding the first programming task from the master test group, which obtained a low survey outcome (see Fig. 5.3). From the text answers it can be concluded that the low outcome was due to a misunderstanding in the initial steps of the tutorial, when a repository with the files should be downloaded. Furthermore, from the sixth row we get a better understanding for the low survey outcome in the Docker section in Fig. 5.10. The PhD test group experienced the information as a "simplification of their intent", which most likely effected the survey outcome. Note that only the PhD and master test groups are present in the text answers in the table. The hobby test group agreed in general to a greater extent with the statements in Fig. 5.3 and Fig. 5.4 and experienced the material as easier and more relevant (see Fig. 5.7 and Fig. 5.5) compared to the other test groups. In the same manner, the text answers given by the hobby test groups was either "No, I don't think so", on the question: *suggestions on improvements* or answers that did not explain the reason for the low survey outcome, in for example section Connect to Wifi in Fig. 5.7.

Table 5.1: Selection of answers from the survey which were considered valuable when analysing the data and identifying future improvements in the tutorial. The text questions the respondents were answering were *Suggestions on improvements* and *If the coding example was not enough, what was missing?* The first text question was present in every survey, while the second only was asked when coding examples were demonstrated. In the first and sixth row we can get a possible explanation to the low survey outcome from the master and PhD test group in Fig. 5.3 and Fig. 5.10 respectively. Note that only the answers from the master and PhD test group is present in the table. The hobby test group agreed to a greater extent with the statements, or experienced the material as more relevant and easier compared with the test groups (see Fig. 5.3 – 5.10. The text answers from the master and PhD test groups where therefore more constructive in its feedback on improvements.

Test group	**Section**	**Text answer**
Master	Programming task 1	It would be good to have a complete set of files for running the program with docker. Since the user might never have used Dockerfiles before and may not have any clue about how to actually compile the code and test it.
Phd	Mutex	This is the biggest jump, skill-wise, in the tutorial - and I completely get that you need to set a healthy balance, and I have no idea how you would make this better, but in my opinion, I think students may get very frustrated with this.
Master	Docker	It is hard to understand the concept if you have never heard of it before. Therefore it would be good to have a example that you could run by yourself beside the tutorial.
Master	Cone detection	I find it easier to have the code example given in the beginning
Master	Code example	Maybe refer to one of the example codes given in the links ,for the user to get familliar with the complete code and where to find different parts. You can write that the user should have the code beside the tutorial just before the code is explained. Refering to line-numbers in the code is as well useful.
Phd	Docker	'The .GITLAB-CI.YML file is a description of how the code should be executed and transformed between different platforms' and DOCKER-COMPOSE replays data into the simulation.' - these statements both have issues. They are wrong, but I feel this is a simplification of their intent
Phd	Task 4	Not sure what this meant: "what will happened if we the code running on the Kiwi car containing prints or plots?

6
Discussion

To answer the first research question regarding how to build the Kiwi car, Figure 5.1 and Figure 5.2 can be observed. The figures illustrate to which extent the manual managed to present the necessary information for assembling the front and back wheels, and if the users understood what they should do in each step. As illustrated in the figures both the information presented and the understanding of the content were given the highest score for the sections that were evaluated. In order to create a common thread though the manual, the approach for which parts that were required and how to assembling them were the same. With this in mind combined with the survey outcome in Figure 5.1 and Figure 5.2, we can presume that the structure of the manual, both when it comes to which parts that were required and the instructions for assembling the pieces will be apprehended. However, the secondary school test group that were supposed to build the Kiwi car guided by the manual was prevented from completing the task due to a change of schedule because of the Corona pandemic restrictions. As a consequence only the initial steps were completed. Furthermore, the steps that were completed, were of a more basic nature and how well the test groups would be able to assemble the more challenging parts, such as the electronics, cannot be concluded.

The programming tutorials in Figure 5.10, shows that the higher expected prior knowledge the test group had, the lower the score was received. This can, on one hand, be interpreted as a positive result to find the minimum requirement to get all the targeted test groups started. On the other hand, it can be interpreted as a disadvantage when it comes to challenging the more experienced programmers. The lower survey outcome from the PhD test group could be explained by their high field expertise. One of the text comments received by the PhD test group, presented in Table 5.1 was concerning the Docker section. The comment criticized the explanation of the .GITLAB-CI.YML file and DOCKER-COMPOSE and mentioned that it was a simplification of their intent. Since the tutorial addressed four test groups, where the PhD test group was expected to have the best prior knowledge, simplifications were considered necessary to decrease complexity and risk of misconceptions. However, the same level of assembly which was applied for the image detection, where each test groups were given the option to pursue a more challenging route, could have been used. Having said that, giving the students too many options may instead make it more complicated. For instance, with more options the students may have to reread the same section to find the level that suits their current knowledge. Having to reread to many times may decrease the motivation and increase the risk of choosing a too easy or difficult level, which in turn may decrease

the motivation further. In the latter case, when the chosen difficulty level was too ambitious, the student might risk misinterpreting the meaning of the section. One further improvement could be to apply level of assembly for a few section where the material is more technically advanced. A simplification of the intent could be presented for the test groups with lower prior knowledge, while the full explanation could be presented for the more experienced groups. In the existing material, external links were supposed to be used to challenge the test groups with better prior knowledge. The collected data was however not sufficient to draw any conclusions regarding how well the external links complemented the information presented in the tutorial.

Figure 5.3 illustrates to which degree the tutorials gave the test groups sufficient information to conduct the task. The result indicates that the overall expression was that they had sufficient information to do the tasks, with the exception for the master test group in the first task. This is probably a consequence of a misunderstanding in the first section of the tutorial where the test groups should have downloaded an existing template, which provides the necessary files to run the program. The only thing the test groups had to do were to implement the Python code. In Table 5.1 the master test group wrote "*It would be good to have a complete set of files for running the program with docker. Since the user might never have used Dockerfiles before and may not have any clue about how to compile the code and test it.*" Most likely the low score for task 1 in Figure 5.3 reflects how the user missed the files rather than the overall content of the tutorial.

On the other hand, if the information to all four test groups was sufficient enough to complete the programming tasks, can be questioned. Only the two test groups with the expected best prior knowledge completed all tasks. If this was a consequence of the time schedule being too tight or if the material was too challenging for the hobbyist and secondary school test groups was not concluded. It could be argued that it was a combination of both of them, but for the secondary school group the main reason for not completing the tutorial was the time constraint. Another observation related to the hobbyist group is that the survey question outcome was above average for the tutorial steps (see Figure 5.10), while the PhD and master test group solved the programming tasks to a greater extent. This may indicate that the tutorials were effective in instructing the user at a specific step but were less effective in providing information on how the steps should be integrated to see the full solution. An alternative explanation could be that the programming task was too challenging for the test groups with expected lower prior knowledge.

To answer the second research question Figure 5.3 and Figure 5.7 can be taken into consideration. The second figure shows that the test groups in general experienced the material as relevant. Furthermore, Figure 5.3 illustrates how most of the test groups experienced that they had sufficient information to do the tasks. Similarly, one can observe in Figure 5.4 that the programming examples were useful for the test groups who developed their own code. Having said that, only the two test groups with the expected highest prior knowledge completed all the tasks. This may indi-

cate that the information presented was not comprehensive enough or too difficult for some of the test groups. Finding the balance of challenging the test groups with the highest prior knowledge without doing it at the expense of the other test groups, could arguably be a key factor during this thesis. Therefore, simplifications have been implemented in the more challenging parts of the theory. One of the more critical text responses in Table 5.1 were regarding simplifications of the intent of .GITLAB-CI.YML file and DOCKER-COMPOSE. A full explanation was considered to be too difficult for students who have barely worked with Docker or Gitlab before. On the other hand, the test groups who work in the field on a daily basis perceived the information as inaccurate. It could be argued that these simplifications are necessary since the project aims to find the minimum requirements for teaching and learning the Kiwi platform and OpenDLV. Therefore, presenting all the information individually for all the test groups is not relevant.

The result from the layout can be seen in Figure 5.9. In general the PhD test group was more critical to the layout in comprehension with the other test groups. The survey outcome was neutral (score = 3) except for the section dealing with Docker, which obtained a lower score (score = 2). This is the same section where the test group was experiencing the information as a simplification of the real intent, which may have affected the overall impression. Regarding the master test group, the survey outcome was going the opposite direction with a high score at the Docker section but then declining. What differs between the sections is the amount of explanatory text in comparison to lists. While the text related to the Docker section is more extensive, but also a simplification of the real purpose, the other sections use one to three sentences to explain the command in question. When it comes to the master test group, it can be seen that the CI/CD and downloading image sections were given lower scores than the other two. The commands for CI/CD and downloading image are both described by single sentences, which potentially can imply that this specific test group was more positive to longer explanations. On the other hand, the possibility that the level of difficulty in the sections affected the survey outcome can not be rejected. More data is required to conclude if the length of the explanation is the reason for the lower outcome in the latter sections.

When it comes to the third research question it can be seen in Figure 5.4 that the coding examples were in general appreciated, while the hints for task three and task four obtained a lower score. Starting with the coding examples, it can be seen that the score is higher for the image detection compared with the messenger, where we set up the communication for the Kiwi car. Two different approaches were used, in the first case for the messengers, a code template was available for the users if they faced problems. For the image detection and recognition the example codes were divided into two programs. The first part contains detecting the cone objects, while the second part focuses on finding their exact positions and estimate the roadside to steer the car. Both programs were running without any implementation from the user. The students were instead supposed to combine these programs to be able to implement the full solution for task three and task four.

Figure 5.4 shows that the coding examples for image detection were more appreciated. One drawback with this solution could arguably be that the test groups may rely too much on the given code, which may decrease their ability to think creatively and find other solutions to the same problem. One of the text comments in Table 5.1 was "*Maybe refer to one of the example codes given in the links, for the user to get familiar with the complete code and where to find different parts. You can write that the user should have the code beside the tutorial just before the code is explained. Referring to line-numbers in the code is as well useful.* " It could be argued that this sentence is illustrating the problem. As mentioned above, it can be discussed that giving a full solution may increase the risk of copying the code as well as reducing the students creativity and impede their problem solving skills.

Regarding the hints at task three and task four the score was low. The only text feedback obtained was related to the question "*What will happened if the code running at the Kiwi car contains prints or plots?* ", where the test groups did not understand the question. No further information concerning the low survey outcome could be deduced from the data.

6.1 Ethics

A potential problem when studying teaching and learning is that the groups under study does not properly reflect the target groups. There are at least three important properties of such groups to keep in mind: Gender, socioeconomic status, and age. Starting with gender, when choosing participants to the four test groups gender equality was taken into consideration. For the postgraduate education (PhD) and undergraduate education (master), each containing one person, one male and one female were chosen. For the hobbyist and secondary school test groups, several persons were involved in building or programming the car. The hobbyist test group consisted mostly of male participants and the person who conducted most of the programming was male. For the secondary school test group, an IT class was selected. This class was chosen as a result of an interest from the teacher who initiated contact with the project, therefore the composition of males and females in the secondary school test group could not be affected. When choosing the participants, the social economical factor was not taken into consideration, but as a result of the material being published online an initiative has been taken to reduce the numbers of expensive components in the car, and hence reduce the price and increase availability. Lastly, when deciding the test groups some constrains were set on the age span of the participators. For instance, most persons in secondary school are between 15 and 19 years old and master and PhD students tends to be in their mid-, late twenties or early thirties. When it comes to the hobbyist test group, the possibility to affect the age of the participator was easier. To increase the age span a thirteen years old participator was chosen. It can be arguable that it would be better to choose an older person to get an ever larger age span, but choosing a young representative for the hobbyist group was consider valuable for this thesis. The reason for this is that the thesis wants to evaluate how well the material was adapted to young persons starting to exploring the field of autonomous vehicles.

6.2 Improvements

The project aims to find the minimum requirements for web content to use the Kiwi platform to teach OpenDLV to the four target groups. The long term goal is to create an online community where anyone with an interest for autonomous vehicles can build their own Kiwi car and start programming it. A first prioritized step is to evaluate how well the manual for building the Kiwi car can guide the user, which this project failed to test. Another critical factor is to reduce the price for the Kiwi car so more people can afford to buy the parts required for building it. Today the cost for one unit is around 4,500 SEK. With a project to be started to reduce the cost, the parts needed to build the car will likely be in the range of 2,000 SEK.

Regarding the programming tutorials there are many areas that need to be improved further before it can target the greater mass. Four areas found to be important are, building a demo website to collect more data, identifying and explaining more sources of error or finding other online sources that can be helpful when troubleshooting a problem, adapt the material better with respect to the four target groups, and expanding the material with more programming examples combined with theory.

6.3 Future work

A possible next step in a future project could be to further extend the target groups with the intention to evaluate if the material is sufficient to guide users from different parts of the world. Since this thesis is a first step in a bigger project which aims to create an online community for autonomous vehicles, translating the material to Spanish or Mandarin may be a suitable first step.

As an ongoing project, a webpage is under development where the material will be published. How the material will be presented in a user friendly way has been taken into consideration when creating the material. Still, modifications will most certainly have to be done when the webpage is completed. As a future project, the material could be modified focusing only on how it can be presented in a more pedagogical way rather than the technical content.

7
Conclusion

This thesis examined how the Kiwi platform can be used to teach OpenDLV to four target groups. The thesis was focusing on both giving instructions for building the Kiwi car but also how to program it. Regarding how well the material guided the user to build the Kiwi car, the data was not extensive enough to draw any general conclusions. However, for the two sections that were evaluated, both the presentation of the information and the instructions were well received.

When programming the Kiwi car, all test groups experienced the information as relevant, while the perceived difficulties differed between the groups. Only the two test groups with the highest expected prior knowledge finished all the steps. If the reason for the other test groups not finishing the steps were due to the time limit or that the material was too advance was not studied. To answer the first research question for all four test groups (secondary education, undergraduate education, postgraduate education and hobbyists) more data would need to be collected from the secondary education test group. Based on the data from the other three test groups, they found the material relevant. The balance between simplifying the explanations, but not simplifying them too much, was the critical factor in the thesis. For the PhD test group, the one with the expected highest prior knowledge, most of the lower survey outcome and negative text comments were related to simplifications. Overall, the test groups agreed with the statement that the tutorials were giving them sufficient information for completing the programming tasks. However, the fact that only the two test groups with the expected highest prior knowledge completed all the tasks indicate that the tutorials may have been too advanced. When presenting the coding example two different approaches were used. When the communication between the microservices were implemented, only a code template was available. For the image detection, on the other hand, the test groups were given the full code. The second approach was more appreciated, but it may be problematic to give away all of the code. It can for instance result in an increased amount of copied code which may harm the development of problem-solving skills. To conclude, the study shows that there is a great interest in the field and that this thesis contributes to motivate and helping students developing their skills while programming the Kiwi car. With this in mind, the thesis has the potential to result in positive effects in the technical development of autonomous systems.

Bibliography

[1] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[2] M. Pompili, G. Serafini, M. Innamorati, F. Montebovi, M. Palermo, S. Campi, H. Stefani, G. Giordano, L. Telesforo, M. Amore, *et al.*, "Car accidents as a method of suicide: A comprehensive overview," *Forensic science international*, vol. 223, no. 1-3, pp. 1–9, 2012.

[3] I. Ratnapala, R. Ragel, and S. Deegalla, "Students behavioural analysis in an online learning environment using data mining," in *7th International Conference on Information and Automation for Sustainability*, IEEE, 2014, pp. 1–7.

[4] S. Liu and M. d'Aquin, "Unsupervised learning for understanding student achievement in a distance learning setting," in *2017 IEEE Global Engineering Education Conference (EDUCON)*, IEEE, 2017, pp. 1373–1377.

[5] C. Berger, B. Nguyen, and O. Benderius, "Containerized development and microservices for self-driving vehicles: Experiences & best practices," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, IEEE, 2017, pp. 7–12.

[6] C. Berger. (). "Chrberger/libcluon," [Online]. Available: `https://github.com/chrberger/libcluon`. (hämtad: 09.06.2021).

[7] P. Davidson and A. Spinoulas, "Autonomous vehicles: What could this mean for the future of transport," in *Australian Institute of Traffic Planning and Management (AITPM) National Conference, Brisbane, Queensland*, 2015.

[8] R. Behringer, S. Sundareswaran, B. Gregory, R. Elsley, B. Addison, W. Guthmiller, R. Daily, and D. Bevly, "The darpa grand challenge-development of an autonomous vehicle," in *IEEE Intelligent Vehicles Symposium, 2004*, IEEE, 2004, pp. 226–231.

[9] J. B. Watson, *Behaviorism*. Routledge, 2017.

[10] M. G. Jones and L. Brader-Araje, "The impact of constructivism on education: Language, discourse, and meaning," *American Communication Journal*, vol. 5, no. 3, pp. 1–10, 2002.

[11] J. Michael, "Where's the evidence that active learning works?" *Advances in physiology education*, 2006.

[12] C. V. L. Plus. (). "The world's largest evidence base on what works best in schools to accelerate student learning," [Online]. Available: `https://www.visiblelearning.com/content/visible-learning-research`. (hämtad: 30.05.2021).

[13] J. Hattie and H. Timperley, "The power of feedback," *Review of educational research*, vol. 77, no. 1, pp. 81–112, 2007.

[14] K. D. Tanner, "Order matters: Using the 5e model to align teaching with how people learn," *CBE—Life Sciences Education*, vol. 9, no. 3, pp. 159–164, 2010.

[15] M. K. Smith, W. B. Wood, W. K. Adams, C. Wieman, J. K. Knight, N. Guild, and T. T. Su, "Why peer discussion improves student performance on in-class concept questions," *Science*, vol. 323, no. 5910, pp. 122–124, 2009.

[16] J. Valverde-Berrocoso, M. d. C. Garrido-Arroyo, C. Burgos-Videla, and M. B. Morales-Cevallos, "Trends in educational research about e-learning: A systematic literature review (2009–2018)," *Sustainability*, vol. 12, no. 12, p. 5153, 2020.

[17] L. Barnard, W. Y. Lan, Y. M. To, V. O. Paton, and S.-L. Lai, "Measuring self-regulation in online and blended learning environments," *The internet and higher education*, vol. 12, no. 1, pp. 1–6, 2009.

[18] K. Steffens, "Self-regulated learning in technology-enhanced learning environments: Lessons of a european peer review," *European journal of education*, vol. 41, no. 3-4, pp. 353–379, 2006.

[19] J.-L. Hung and K. Zhang, "Revealing online learning behaviors and activity patterns and making predictions with data mining techniques in online teaching," *MERLOT Journal of Online Learning and Teaching*, 2008.

[20] chrberger. (). "Chalmers-revere/opendlv," [Online]. Available: `https://github.com/chalmers-revere/opendlv`. (hämtad: 09.06.2021).

[21] M. Fowler. (). "Microservices," [Online]. Available: `https://martinfowler.com/articles/microservices.html`. (hämtad: 09.06.2021).

A

Survey questions

A.1 Survey for buildning the car

For the manual the questions were repeating the same for all the tasks. The first two questions should be answered with a number and the third with text.
The questions were

The manual provided enough information to perform the specified steps

Totally disagree (1) (2) (3) (4) (5) Totally agree

It was possible to understand what was to be done

Totally disagree (1) (2) (3) (4) (5) Totally agree

(Voluntary) Other considerations or improvement measures

A.2 Survey for programming the car

The sections below will present the questions related to the task described in the tutorial. Since the questions are not the same for the different tasks, the headline on the section will correspond to the task in the tutorial.

A.2.1 Introduction

The links and tutorials guide the user to download the necessary programs

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the material?

Totally disagree (1) (2) (3) (4) (5) Totally agree

How relevant was the material from your interests?

Totally disagree (1) (2) (3) (4) (5) Totally agree

Suggestions on improvements

A.2.2 Booting the system

The instructions and tutorial for booting the systems is useful for understanding how to start them.

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the material?

Totally disagree (1) (2) (3) (4) (5) Totally agree

How relevant was the material from your interests

Totally disagree (1) (2) (3) (4) (5) Totally agree

Suggestions on improvements

A.2.3 Connect to wifi

The instructions and tutorial is guiding the user do connect to the Wifi

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the material?

Totally disagree (1) (2) (3) (4) (5) Totally agree

How relevant was the material from your interests

Totally disagree (1) (2) (3) (4) (5) Totally agree

If you used the "source of error". It was useful to solve or identify what may cause the problem

Totally disagree (1) (2) (3) (4) (5) Totally agree

Suggestions on improvements

A.2.4 Control the robot

The instructions and tutorial is guiding the user how to access the interface

Totally disagree (1) (2) (3) (4) (5) Totally agree

m

The instructions and tutorial is guiding the user how to steer the robot with the joystick.

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the material?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the material from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

If you used the "source of error". It was useful to solve or identify what may cause the problem (Voluntary)

Totally disagree (1) (2) (3) (4) (5) Totally agree

Suggestions on improvements

A.2.5 Code example

The coding example was helpful when developing the code

Totally disagree (1) (2) (3) (4) (5) Totally agree

The number of instructions was enough to get a hint of what should be done, but not to many to remove the students ability to solve the task on their own

Totally disagree (1) (2) (3) (4) (5) Totally agree

If the coding example was not enough, what was you missing?

How advanced did you experience the material?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the material from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.6 Why do we use docker? - Compiling code with docker

The introduction to why we use docker motivates why we using it when building our programs

Totally disagree (1) (2) (3) (4) (5) Totally agree

The purpose of the three docker commands is clear

Totally disagree (1) (2) (3) (4) (5) Totally agree

Links to further explore docker on your own is helpful. If you did not use the external links, skip this question.

Totally disagree (1) (2) (3) (4) (5) Totally agree

The tutorials in the section is helpful for understanding the content

Totally disagree (1) (2) (3) (4) (5) Totally agree

The layout of mixing text, textboxes, lists and tutorial gives a good overview and helps the reader identify the most important steps.

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the material?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the material from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.7 Programming task 1

The information presented in the previous section is enough to understand what to do in the task

Totally disagree (1) (2) (3) (4) (5) Totally agree

The task is in the line with the information presented in the previous sections

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the task?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the task from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.8 Git and branches

The information and tutorial helps the student to understand for pushing code and working in branches

Totally disagree (1) (2) (3) (4) (5) Totally agree

The layout of mixing text, textboxes, lists and tutorial gives a good overview and helps the reader identify the most important steps

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the material?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the material from your interests?

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.9 CI/CD

The brief introduction to CI/CD gives a basic understanding why we are using it

Totally disagree (1) (2) (3) (4) (5) Totally agree

The links gives the user the possibility to get a deeper understanding of the topic

Totally disagree (1) (2) (3) (4) (5) Totally agree

The layout of mixing text, textboxes, lists and tutorial gives a good overview and helps the reader identify the most important steps

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the material?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the material from your interests?

Not relevant (1) (2) (3) (4) (5) Very relevant

If you used the "source of error". It was useful to solve or identify what may cause the problem (Voluntary)

Totally disagree (1) (2) (3) (4) (5) Totally agree

Suggestions on improvements

A.2.10 Docker login/download/run

The instructions and tutorial is guiding the user how to login, download and run the program.

Totally disagree (1) (2) (3) (4) (5) Totally agree

The layout of mixing text, textboxes, lists and tutorial gives a good overview and helps the reader identify the most important steps

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the material?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the material from your interests?

Not relevant (1) (2) (3) (4) (5) Very relevant

If you used the "source of error". It was useful to solve or identify what may cause the problem

Totally disagree (1) (2) (3) (4) (5) Totally agree

Suggestions on improvements

A.2.11 Programming task 2

The information presented in the previous section is enough to understand what to do in the tas

Totally disagree (1) (2) (3) (4) (5) Totally agree

The task is in the line with the information presented in the previous sections

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the task?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the task from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.12 Data replay

The links and tutorials guide the user how to work with data replay

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the task?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the task from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.13 Terminal input and threads

The coding example was helpful when developing the code

Totally disagree (1) (2) (3) (4) (5) Totally agree

The number of instructions was enough to get a hint of what should be done, but not to many to remove the students ability to solve the task on their own

Totally disagree (1) (2) (3) (4) (5) Totally agree

If the coding example was not enough, what was missing?

How advanced did you experience the task?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the task from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.14 Cone detection

The coding example was helpful when developing the code

Totally disagree (1) (2) (3) (4) (5) Totally agree

The number of instructions was enough to get a hint of what should be done, but not to many to remove the students ability to solve the task on their own

Totally disagree (1) (2) (3) (4) (5) Totally agree

If the coding example was not enough, what was missing?

The key concept of dilation, erosion and kernels could be understood with the information presented

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the task?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the task from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.15 Cone position

The coding example was helpful when developing the code

Totally disagree (1) (2) (3) (4) (5) Totally agree

The number of instructions was enough to get a hint of what should be done, but not to many to remove the students ability to solve the task on their own

Totally disagree (1) (2) (3) (4) (5) Totally agree

If the coding example was not enough, what was missing?

The links gives the user the possibility to get a deeper understanding of the classes, attributes and methods used in the examples

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the task?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the task from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.16 Programming task 3

The information presented in the previous section is enough to understand what to do in the task

Totally disagree (1) (2) (3) (4) (5) Totally agree

The task is in the line with the information presented in the previous sections

Totally disagree (1) (2) (3) (4) (5) Totally agree

The hints were useful

Totally disagree (1) (2) (3) (4) (5) Totally agree

The hints exposed to many problems that the student could figure out on their own

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the task?

Very easy (1) (2) (3) (4) (5) Very hard

Did you use the ode4Session or VideoCapture to replay the data?

◯ ode4Session ◯ VideoCapture ◯ Other

How relevant was the task from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

A.2.17 Programming task 4

The information presented in the previous section is enough to understand what to do in the task

Totally disagree (1) (2) (3) (4) (5) Totally agree

The task is in the line with the information presented in the previous sections

Totally disagree (1) (2) (3) (4) (5) Totally agree

The hints were useful

Totally disagree (1) (2) (3) (4) (5) Totally agree

The hints exposed to many problems that the student could figure out on their own

Totally disagree (1) (2) (3) (4) (5) Totally agree

How advanced did you experience the task?

Very easy (1) (2) (3) (4) (5) Very hard

How relevant was the task from your interests

Not relevant (1) (2) (3) (4) (5) Very relevant

Suggestions on improvements

B

All answers

All the collected data is presented below. Note that some data was received after the completion of the report and is therefore not included in the result section. None of the excluded results differs significantly from the result presented in the result section.

Table B.1: All the graded answers

Question	Hobbiest	Master	Phd	Scale
Introduction				
1	5	5	5	disagree/agree
2	1	4	2	disagree/agree
3	5	5	5	disagree/agree
Booting the system				
1	5	5	5	disagree/agree
2	1	2	1	disagree/agree
3	4	5	5	disagree/agree
Connect to wifi				
1	5	4	4	disagree/agree
2	1	4	2	disagree/agree
3	2	5	5	disagree/agree
4	-	5	-	disagree/agree
Control the robot				
1	5	5	4	disagree/agree
2	5	5	4	disagree/agree
3	1	5	2	very easy/hard
4	4	1	5	not/very relevant
5	-	-	-	disagree/agree
Code example				
1	5	4	4	disagree/agree
2	5	4	3	disagree/agree
4	3	3	2	very easy/hard
5	5	5	5	not/very relevant
Why do we use docker? - Compiling code with docker				
1	5	5	2	disagree/agree
2	4	5	2	disagree/agree
3	-	5	3	disagree/agree

Continued on next page

Table B.1 – continued from previous page

Question	Hobbiest	Master	Phd	Scale
4	4	4	2	disagree/agree
5	5	5	2	disagree/agree
6	2	4	4	very easy/hard
7	5	5	5	not/very relevant
Programming task 1				
1	5	2	4	disagree/agree
2	5	5	5	disagree/agree
3	2	5	4	very easy/hard
4	5	5	5	not/very relevant
Git and branches				
1	5	4	4	disagree/agree
2	5	5	3	disagree/agree
3	4	3	3	very easy/hard
4	4	5	5	not/very relevant
CI/CD				
1	-	5	1	disagree/agree
2	-	5	3	disagree/agree
3	-	4	3	disagree/agree
4	-	4	3	very easy/hard
5	-	5	5	not/very relevant
6	-	-	-	disagree/agree
Downloading the image to the Kiwi car				
1	-	4	5	disagree/agree
2	-	3	4	disagree/agree
3	-	4	3	very easy/hard
4	-	5	5	not/very relevant
5	-	3	5	disagree/agree
Programming task 2				
1	-	5	4	disagree/agree
2	-	5	3	disagree/agree
3	-	2	4	very easy/hard
4	-	5	5	not/very relevant
Data replay				
1	-	5	4	disagree/agree
2	-	1	4	very easy/hard
3	-	5	5	not/very relevant
Terminal input and threads				
1	-	4	5	disagree/agree
2	-	5	2	disagree/agree
4	-	3	5	very easy/hard
5	-	5	5	not/very relevant
Cone detection				

Continued on next page

Table B.1 – continued from previous page

Question	Hobbiest	Master	Phd	Scale
1	-	5	5	disagree/agree
2	-	5	5	disagree/agree
4	-	5	4	disagree/agree
5	-	2	5	very easy/hard
6	-	5	5	not/very relevant
Cone position				
1	-	5	4	disagree/agree
2	-	5	3	disagree/agree
4	-	5	3	disagree/agree
5	-	2	5	very easy/hard
6	-	5	5	not/very relevant
Programming task 3				
1	-	5	5	disagree/agree
2	-	5	5	disagree/agree
3	-	5	5	disagree/agree
4	-	2	3	disagree/agree
5	-	4	4	very easy/hard
7	-	5	5	not/very relevant
Programming task 4				
1	-	4	5	disagree/agree
2	-	5	5	disagree/agree
3	-	4	4	disagree/agree
4	-	1	4	disagree/agree
5	-	4	4	very easy/hard
6	-	5	5	not/very relevant

Table B.2: All the text answers

Test group	Text answer
Introduction	
Hobbiest	It was not very clear that we should clone files from Gitub but I understood that with a little time of thinking in the beginning of step 5.
Master	-It says: "sudo apt-get install socat installs cmake" :) i guess it is a typo -I think it is good to explain what Cmake is -If the tutorial should be for students, then Gitlab should be logged in with chalmers-id. Then it might be good to mention.

Continued on next page

Table B.2 – continued from previous page

Test group	Text answer
Phd	As my work has been essentially doing this the last 6 months, I could entirely skip this section. However, after reading it, I can say that it would work quite well for a new student, as a linear tutorial - Do this, do this, do this, etc. However, I feel that the student will likely not understand why certain things are being done and may feel slightly overloaded.
Booting the system	
Hobbiest	It should be interesting to know how to shut down the car correctly.
Master	-Maybe mention that they should leave the car while calibrating and mention that it takes a while to calibrate and if possible, why they need to do the steps. -Nice homemade videos!
Phd	Quite easy to follow. Well done.
Connect to wifi	
Hobbiest	No, I don't think so.
Master	Maybe divide the section into code and text to make it clearer. The website explaining how to connect to wifi could be experienced very advanced since not all steps are described. It looks like you can click on the addresses to the beaglebone and raspberry.
Phd	The main concern I have is the 'passwords do not work' section of the source of error - you haven't provided a solution, but instead just explained why they may not work. How would you fix this if the default passwords do not work?
Control the robot	
Hobbiest	I have one small improvement when downloading the recording from the camera. It would be easier to understand witch video I just record if the latest one came first in the list.

Continued on next page

Table B.2 – continued from previous page

Test group	Text answer
Master	Maybe it is possible to show both camera- and joystic-window at the same time in the video.
Phd	This is good and easy enough to follow. The pdf I have doesn't have pictures, but if there aren't any, maybe consider putting in a few?

Code example

Hobbiest (Question 3)	-
Master (Question 3)	Maybe explain what is cid.
Phd (Question 3)	The code was enough, but it felt like a 'Do this' instead of 'We use this, because it does this', which I feel would enable the students to experiment more with the functions.

Hobbiest	No, I don't think so.
Master	It is a bit confusing that the codes comes twice in 5.2. Maybe refer to one of the example codes given in the links ,for the user to get familiar with the complete code and where to find different parts. You can write that the user should have the code beside the tutorial just before the code is explained. Referring to line-numbers in the code is as well useful.
Phd	Being too concise with explanations of code blocks is one of the biggest issues in software engineering documentation - Do not be afraid to spell out exactly what a function does.

Why do we use docker? - Compiling codewith docker

Continued on next page

Table B.2 – continued from previous page

Test group	Text answer
Hobbiest	- The text would be more helpful if you describe why we use the "tag" in the Docker-commands. Before this project I start looking at Docker and because of that I understood this quickly. - Even if I have test Docker before, I tested to build a Python script and run it with Docker to know everything is alright. The terminal said about I didn't had access to Docker daemon.* I search on Google to find out how to fix this. I upon that I had to add myself into the Docker group by first typing in the terminal "sudo usermod -a -G docker lage" (lage is my user) and after that "newgrp docker". Lastly I had to log out and in again but after that it worked great to build containers. * I have not stored the error. When I solved this problem I had to log out and in. Then all going programs closed and also the terminal. is as well.
Master	It is hard to understand the concept if you have never heard of it before. Therefore it would be good to have a example that you could run by yourself beside the tutorial. Maybe it would be good to explain what a container is as well.
Phd	'The .gitlab-ci.yml file is a description of how the code should be executed and transformed between different platforms' and 'docker-compose replays data into the simulation.' - these statements both have issues. They are wrong, but I feel this is a simplification of their intent. It would benefit the students a lot more to word it like '.gitlab-ci is the file that holds the instructions for the automated tools hosted on gitlab that we will later use to validate our builds on different platforms', and 'the docker-compose file is used to chain docker instructions together, and in this case, we will use it to build our simulation environment and replay data using prebuilt docker images', or something along these lines.

Programming task 1

Hobbiest	No, I don't think so.

Continued on next page

Table B.2 – continued from previous page

Test group	Text answer
Master	It would be good to have a complete set of files for running the program with docker. Since the user might never have used Dockerfiles before and may not have any clue about how to actually compile the code and test it.
Phd	As a problem task, this is completely in line with the tutorial direction, however I think that some students may struggle with the overcomplexities presented with the docker segment prior to this.

Git and branches

Hobbiest	First I did not understood that I should create a new repository and clone the code from that repository.
Master	Maybe explain how to setup the repository and merge branches.
Phd	-

CI/CD

Hobbiest	-
Master	Maybe have a gitlab-ci.yml example that works with a example code just to demonstrate and give the user a starting point. I would not have been able to create the file if i hadn't had some knowledge before.
Phd	'For more information about CI/CD click here' is not a sufficient explanation on what CI/CD is or why we use it. Also, this is just a language semantic, but acronyms should be proceeded by the full noun on the first usage, and not the other way around.

Downloading the image to the Kiwi car

Hobbiest	-
Master	Just mention that the registry address might be different
Phd	Fairly straightforward.

Programming task 2

Continued on next page

Table B.2 – continued from previous page

Test group	Text answer
Hobbiest	-
Master	suggesting the user to make a Docker build to test for errors before building at git and downloading to the kiwi
Phd	I feel that some sort of positive feedback here for the student would be appreciative. This is a major milestone in the tutorial and it feels a little underwhelming.

Data replay

Hobbiest	-
Master	Somehow i could not just copy the terminal lines on step 2, but i could then just copy the link and manually write the rest of the command.
Phd	'This commands allow us to get access to the running servers and need to be retyped every time Ubuntu is restarted.' - It's probably easier to say 'This command allows our programs to access the internal GUI server built into ubuntu, and that internal server gives us our images when we replay the data'. It explains it a little better, as well as provides context for the students who (in my experience) forget to do this the most.

Terminal input and threads

Hobbiest (Question 3)	-
Master (Question 3)	-
Phd (Question 3)	-

Hobbiest	-
Master	-
Phd	This is the biggest jump, skill-wise, in the tutorial - and I completely get that you need to set a healthy balance, and I have no idea how you would make this better, but in my opinion, I think students may get very frustrated with this.

Continued on next page

Table B.2 – continued from previous page

Test group	Text answer
Cone detection	
Hobbiest	-
Master	I did not follow how the middle left position became 1 with the given kernel in the picture example.
Phd	This is very well done and more than enough for the students. Good job
Hobbiest (Question 3)	-
Master (Question 3)	-
Phd (Question 3)	-
Cone position	
Hobbiest	-
Master	Got nothing :)
Phd	'By importing polynomial we can fit a n-degree polynomial function to our data points.' - For 75+% of the students going into this as an introduction to robotics, this sentence is going to be completely alien. I would suggest simplifying it - you can be very explicit with what each word means.
Hobbiest (Question 3)	-
Master (Question 3)	Maybe an explaination that the code won't show anything at first
Phd (Question 3)	-
Programming task 3	
Hobbiest	-
Master	-

Continued on next page

Table B.2 – continued from previous page

Test group	Text answer
Phd	Mine was od4 in c++ but the cv2 programming is the same. From what I can see, your tutorial works quite well for python and I cannot find any major python issues
Hobbiest (Question 6)	-
.	
Master (Question 6)	VideoCapture
Phd (Question 6)	ode4Session
Programming task 4	
Hobbiest	-
Master	Not sure what this meant: "what will happened if we the code running on the Kiwi car containing prints or plots?" How should the user evaluate the results?
Phd	what will happened if we the code running on the Kiwi car containing prints or plots?' - I am not sure what you mean by this sentence.

C

Full sections name

Table C.1: Corresponding figure index and sections in tutorial

Name in tutorial	**Name in figures**
Setting up the environment	Set up Ubuntu
Booting the system	Booting system
Connect to Wifi	Connect to Wifi
Control the robot	Control robot
Writing the first code	First code
Why do we use docker?	Docker
First programming task	1st task
Push and compile code on Gitlab	Push code
Continuous integration and deployment	CI/CD
Downloading the image to the Kiwi car	Download image
Second programming task	2nd task
Data replay	Data replay
Threads and mutex	Mutex
Cone detection	Cone detection
Find position of the cones	Cone position
Task 3	task 3
Task 4	task 4

D

Manual

This document will provide step by step instructions for how to build a Kiwi car. It will be divided into five section, namely

1. Risk analyzes for human and robot
2. Component list
3. Instruction for how to build the chassis
4. Integration of technical systems such as camera and sensors.
5. Start up the Kiwi car.

The documentation aims to reach students with different preknowledge. Therefore the instruction will be of basic nature and sections where we are going deeper into the details will be marked with *. These are not required for building the Kiwi car but will give you an extra insight into how the technical systems works.

Contents

1 Risk analyzes for human and robot

Before starting to assemble the robot pay close attention to the safety description. Not understanding the safety description may result in injury.

The battery is a 7.4V lithium-ion polymer battery. When working with the robot make sure that the main power connectors are separated. These connectors can be seen in figure 1. Also pay attention to the instruction on the backside of the battery.

Figure 1: Before working with the battery disassemble the two main connectors.

Before assembling any part it is important to secure the daughter board. In figure 16 you can see that the daughter board has a switch on the right side. The switch controls the power from the battery and its wires need to be secured. In figure 16 the red tape is covering the wires.

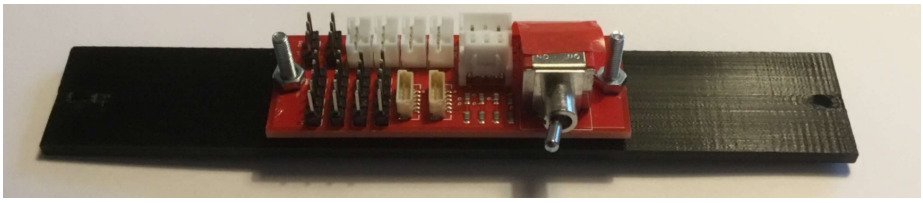


Figure 2: Daughter board fixated on the roof bridge.

Regarding the robot, it has some weak spots that the user needs to be aware of. Between the two front wheels the servo is located. If the steering wheels are forced to its extreme position, either turning left or right, the servo may not be able to turn the wheels enough. This in a consequence of the construction of the steering link system. Still the servo tries to increase the torque by increasing the current, which will result in a cracking sound. To avoid burning the circuits, turn off the Kiwi car according to the instruction presented in section 5.1.

When charging the Kiwi car one has to be careful. The battery connector is located at the Beagleboard chip on the right side of the car. One can see the connector in the lower end of the chip in figure 35. Next to the battery connector there is another cord connecting the two chips. The attachment to the chips is weak and very sensitive for displacement.

2 Component list

The components needed to build the Kiwi car is listed below.

Name	Location	Figure number	3D printed	Quantity
Base plate	Base	44	Yes	1
Wheels	Front and back	45	Yes	4
Rim	Inside wheels	47	Yes	4
Distance	Inside front wheels	46	Yes	4
Tyre brackets	Front wheels	48	Yes	2
Wheel shaft	Front and back	49	Yes	2
Cup joint	Between wheel shaft and wheels	50	Yes	2
Screw (4.3x21)		51	No	52
Screw (2.3x13)	At motor holder	51	No	6
Screws (3.6x11.3)	At camera holder and steering frame	-	No	8
Screw nut (inner diameter 2.7mm)		51	No	52
Screw nuts (inner diameter 1.7mm)		-	No	4
Screw washer (inner diameter 2.7mm)		-	No	8
Right wheel holder up	Front wheels	52	Yes	1
Left wheel holder up	Front wheels	52	Yes	1
Right wheel holder down	Front wheels	53	Yes	1
Left wheel holder down	Front wheels	53	Yes	1
Bearing(5x10x4mm)	Front wheels	54	No	4
Bearing(4x8x3mm)	At transfer pin	54	No	1
Bearing(8x12x3.2mm)	At gear	54	No	2
Link	Between front wheels	55	Yes	1
Steering link	Between front wheels	56	Yes	1
Hollow ball	At servo horn	51	No	2
Front bumper	Front	57	Yes	1
Rear bumper	Back	58	Yes	1
Steering frame	Front	59	Yes	1
Brushless motor and ESC *	Center	60	No	1
Motor holder	Center	61	Yes	1
Motor housing	Center	62	Yes	1
Shaft connector	Ahead of battery	63	Yes	1
Transfer pin	Between shaft connector and gear	64	Yes	1
Gear	Between front wheels	64	No	1
Differential housing	Between front wheels	65	Yes	1
Glue	Below ESC	-	No	1
Servo	Between front wheels	67	No	1
Servo horn	At the servo	68	Yes	1
RaspberryPi holder	Right side of the car	69	Yes	1
Raspberry Pi	Right side of the car	70	No	1
Sd card	At Raspberry Pi	71	No	1
Beaglebone	Left side of the car	72	No	1
Beaglebone holder	Left side of the car	73	Yes	1
Side bracket	Left and right side of the car	74	Yes	2
Roof bridge	On top of side brackets	75	Yes	2

Table 1: Component list for the Kiwi car. * ESC stands for electric speed controller.

Name	Location	Figure number	3D printed	Quantity
Daughter board	Below front roof bridge	77	No	1
Battery (7.4V lithium-ion polymer)		78	No	1
Straps	On battery	-	No	2
Lipo cable	Between battery, daughter board, Beaglebone	83	No	2
Pd connector outlet	Battery and Beagleboard	79	No	1
Pd connector inlet	Control center and Battery	79	No	1
Jst inlet (4.9mm long 4.8mm wide)	Beagleboard and control center	80	No	2
Jst inlet (4.9mm long 6.8mm wide)	Beagleboard and control center	80	No	3
Crimp housing	At Beagleboard	81	No	9
LiYv wires (1.1mm in diameter)	At Beagleboard and IR sensors	82	No	3
Jst wires (0.7mm in diameter)	IR sensors	82	No	11
Wifi antennas	Base plate	84	No	2
Raspberry Pi camera	At front roof bridge	85	No	1
Picam holder	At front roof bridge	86	Yes	1
Usb cable	Between Raspberry Pi and Beagleboard	87	No	1
Ultrasonic Range Finder -I2C	Front and Back	88	No	2
Sensor holder front	Front	89	Yes	1
Sensor holder back	Back	90	Yes	1
Ribbon cable	Between Ultrasonic and control center	91	No	2
IR sensor	Left and right front side bracket	92	No	2

Table 2: Component list for the Kiwi car. * ESC stands for electric speed controller.

3 Building the chassis of the car

This section aims to give instructions for how to build the chassis of the car. How to integrate the sensors and chips will be explained in section 4.

3.1 Back wheels

Starting with the back wheel systems one needs

- 1 wheel
- 1 rim
- 1 wheel shaft
- 1 cup joint
- 1 tire bracket
- 1 screw (4.3x21)
- 1 screw nut

for each system. After assembling the pieces the system will look similar to figure 3.

Figure 3: Illustration of the front wheels.

Place the rim inside the wheel. Then place the cup joint inside the tire bracket before putting a screw through the top of the cup joint and through the rim. To secure that the screw is fixed, place a screw nut at the bottom of the screw. Next place the wheel shaft at the top of the cup joint as illustrated in figure 3. Repeat the steps for the second front wheel.

Answer question at https://docs.google.com/forms/d/e/1FAIpQLSdfbx1GMiT4CLncOOk9d99mOmVWhpaRJw9JwqLwPtVCy586

3.2 Front wheels

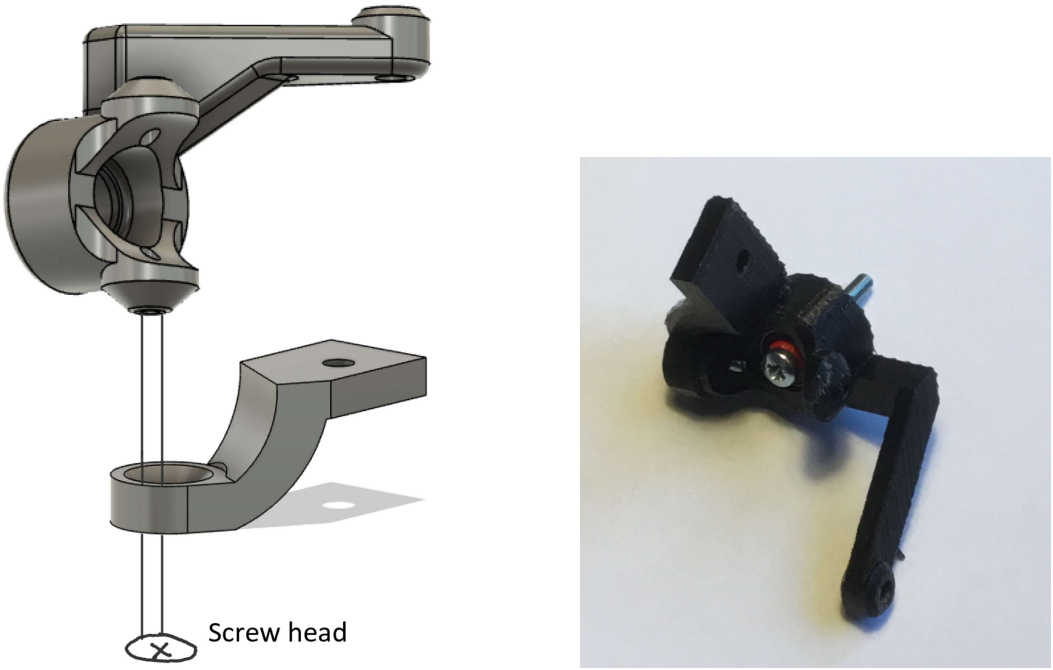
For the front wheels the implementation of the steps for the left and right wheel is identical but due to non symmetrical components for the two wheels one has to be careful. Since the implementation is identical, only the instructions for the right wheel will be given. In case the parts differ between the left and right wheel the reader will be notified. The corresponding part for the left wheel will be given within parentheses, for example (*corresponding part for left wheel*). After assembling the pieces the front wheel will look similar to figure 4.

Figure 4: Illustration of the right front wheel.

For one front wheel we need

- 1 wheel
- 1 rim
- 1 right wheel holder up (*left wheel holder up*)
- 1 right wheel holder down (*left wheel holder down*)
- 2 bearing(5x10x4)
- 2 distance
- 1 screw (4.3x21)
- 1 screw (3.6x11.3)
- 1 screw nut

Place the rim inside the wheel. Next fix the right wheel holder up (*left wheel holder up*) and the right wheel holder down (*left wheel holder down*) by a 3.6x11.3mm screw. Figure 5a illustrates how the right wheel holders should be put together.



(a) (b)

Figure 5: a) Right wheel holder with the direction of the screw b)Illustration of right front wheel.

Next merge the distances and the bearings. Place one of them on the screw so that the distance is facing the head of the screw (4.3x21). Place the screw with the bearing in the wheel holder as shown in figure 5b. Thread on the second bearing on the screw, also letting the bearing facing the top. Fixate the wheel and the wheel holder together with a screw and put the screw nut on the inside of the rim.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSc65H9xOsxuHJ2-EyqQ_jWoyn6tjTxpRMd7BqWG2m0rXCu`

3.3 Link between the wheels

For the link, one needs

- 1 link
- 1 steering link
- 1 screw (4.3x21)
- 1 hollow ball
- 1 screw nut

After assembling the pieces the system will look similar to figure 6.

Figure 6: Link system for the front wheels.

Thread the hollow ball and the link on the screw. Merge the link and the steering link together and fix it with a screw nut.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSdjXhY-CxuhcamPZ7-Z4uxMnZy8f6rGWVGhK39Ej71Ny9m`

3.4 Bottom plate

After assembling the pieces the chassis will look similar to figure 7. Note that the car in the figure has some extra components and wires, such as sensors and wifi amplifier. We will return to how to fix these components and wires at a later stage.

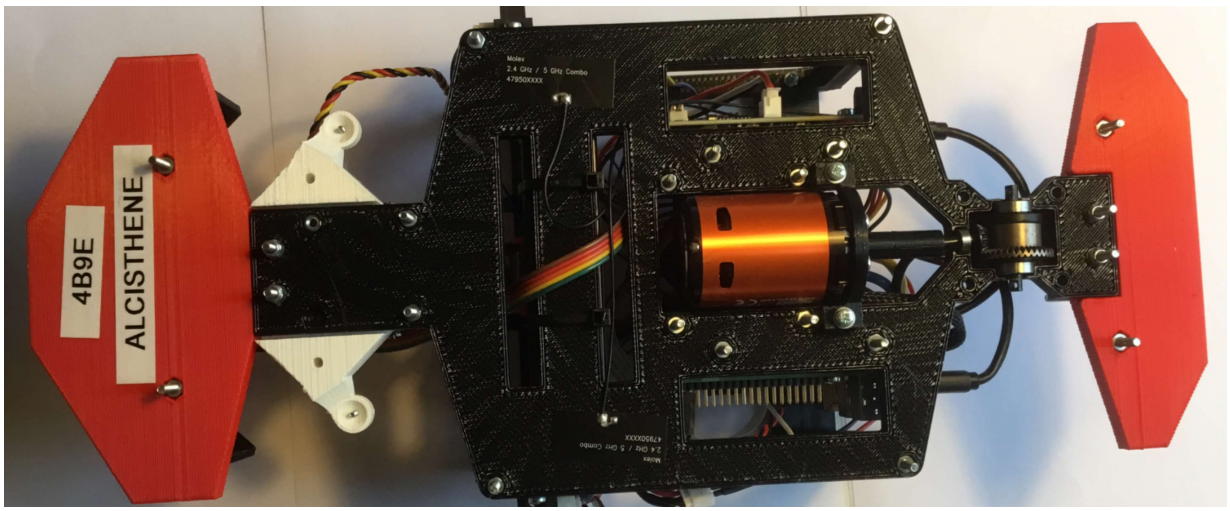


Figure 7: Illustration of the ground plate of the car.

For the bottom plate one needs

- 1 base plate
- 1 front bumper
- 1 rear bumper
- 1 steering frame
- 1 brushless motor including battery and electric speed controller
- 1 motor holder
- 6 screws (2.3x13)

- 1 motor housing
- 1 shaft connector
- 8 screw (4.3x21)
- 8 screw nut

Starting with the base plate. The base plate is shown in figure 8 and the bottom plate, which for instance include the base plate and front and rear bumper, is shown in figure 7.

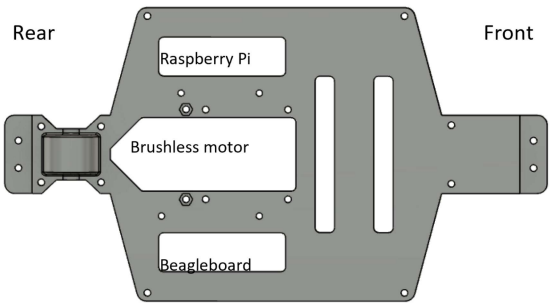


Figure 8: Illustration of the base plate of the car.

Identify the recess in the rear bumper (the smaller of the two bumpers) and place it at the rear end of the base plate. Fix the two pieces with two screws and two nuts. Repeat the process for the front bumper. Place the steering frame so its front holes match with the holes on both the front bumper and the base plate. Fix one of the screws and make sure that the holes in the back of the steering frame match with the ones on the base plate before fixing the three remaining screws and nuts.

Flip the bottom plate, so that the underside is facing downwards. Identify the gap where the brushless motor should be located which can be seen in figure 8. In the upper left and right corner of the gap, with respect to the cars orientation, there are two screw holes. Match these holes with the holes on the motor housing. You will notice that the motor housing has an extra two screw holes, which match the holes of the base plate. Fix the motor housing with four screws and four nuts.

The engine consist of two pieces, namely the brushless motor and the electronic speed control (ESC). First we are going to deal with the brushless motor and in the end of this section we are going to return to the electronic speed control. In figure 9 one can see the set up for the brushless motor, when the pieces are assembled. Place the motor holder with six screws (2.3x13) on the top of the battery. Notice that the orientation of the brushless motor holder can't be chosen arbitrarily. Instead make sure that you can assemble the motor holder with the base plate so that the connectors going out from the brushless motor is facing upwards, against the roof of the car. Fixate the motor holder and place the shaft connector on the metal pin on the top of the brushless motor. Going back to figure 8 we saw that the top two screw holes next to the battery gap, with respect to the orientation of the car where intended for the differential housing. Instead fix the motor holder in the third position.

Figure 9: Brushless motor with motor holder and the shaft connector.

Now we want to transfer the torque from the motor to the back wheels. To do so, we need

- 1 transfer pin (171x11mm)
- 1 gear (340x233mm)
- 1 bearing(4x8x3mm)
- 2 bearing(8x12x3.2mm)
- 1 differential housing
- 4 screw (4.3x21) (if the differential housing is being fixed)
- 4 screw nut (if the differential housing is being fixed)

Locate the smallest bearing at the transfer pin. At the top of the shaft connector there is a small hole with the same pattern as the transfer pin. Connect the transfer pin and the shaft connector. Alongside the shaft connector and the transfer pin, there is a cavity. To increase the stability of the system adjust the bearing on the transfer pin to fit the cavity. The correct position can be seen in figure 10. Connect the transfer pin to the gear according to figure 10. At both sides of the gear there is an attachment pin, place one of the larger bearings on either side. The chassis will now look similar to figure 7. If you are planning to continue with the integration of the back wheels, don't immediately fix the differential housing. If not, place the differential housing over the gear and fix it with four screws and four nuts.

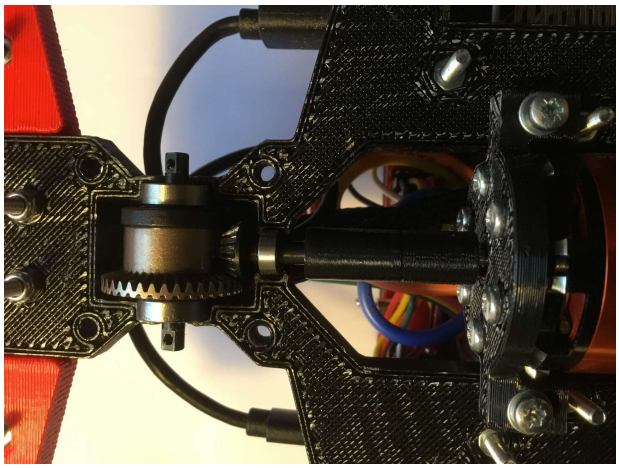


Figure 10: Position of the metal ring on the transfer pin.

Lastly we will fixate the electronic speed controller (ESC). We need

- 1 brushless motor and electric speed controller
- 1 battery base T-connector system
- glue

When the electronic speed control is in the correct position, the central part of the car seen from above will look similar to figure 11. Glue the ESC on top of the motor holder with its colourful wires facing backward as in figure 11. The second and fourth wire counting from the left will later on be connected to the battery. To prepare for this, merge the red wire with the plus port of the base T-connector system and the black wire with the minus port. The plus port is marked with a plus sign (+) and the minus port with a minus sign (-) on the base T-connector system.

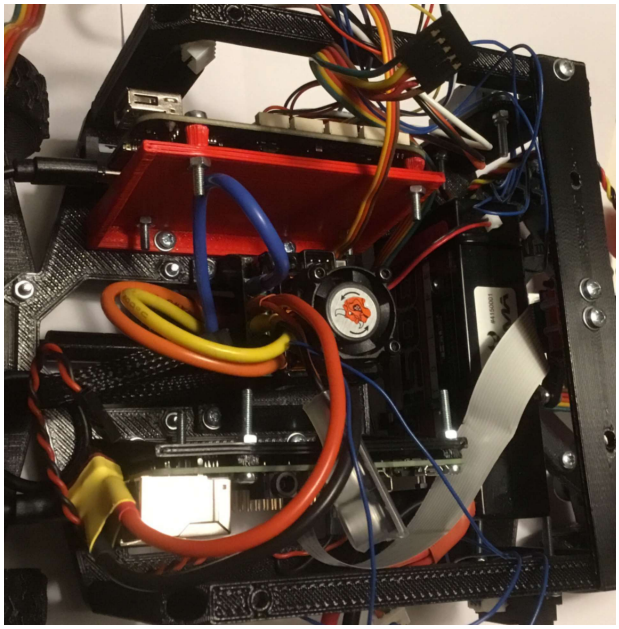


Figure 11: Central part of the car seen from above

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSebON_LnAzasBZsY07oqYVX_89bopFlrS6-JQ1oyHObTYC`

3.5 Integrate the wheels

Starting with the back wheels we need

- 1 base plate from section 3.4
- 2 back wheel systems from section 3.1
- 4 screws (4.3x21)
- 4 screw nuts

assuming that your Kiwi car look similar to figure 7, where the differential housing is not yet fixed. Grab the back wheel system as shown in figure 3. The wheel system must be fixed on the underside of the differential housing. Make sure that the wheel shaft is connected to the pin sticking out from the side of the differential housing and that the tire bracket lies evenly to the differential housing. Fix the tire bracket with two screws and two nuts. Repeat these steps for the second wheel system.

For the front wheels one needs

- 1 right front wheel system from section 3.2
- 1 left front wheel system from section 3.2
- 1 link system from section 3.3
- 1 servo
- 1 servo horn
- 4 screw (4.3x21)
- 2 screw (3.6x11.3)
- 1 hollow ball
- 4 screw nuts

The front wheel system has three screw holes. Starting with the smallest hole in the middle and place it below the extension on the steering frames side. How this should be done can be seen in figure 12. Then match the wheel holder down to the screw hole in the steering frame and fixate them. Repeat the same steps for the second front wheel.

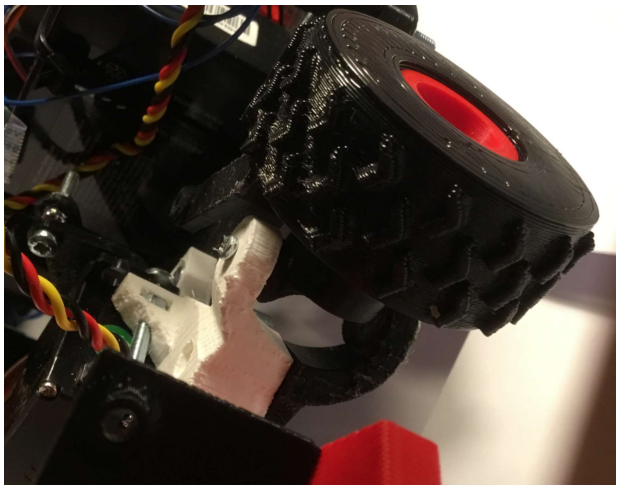
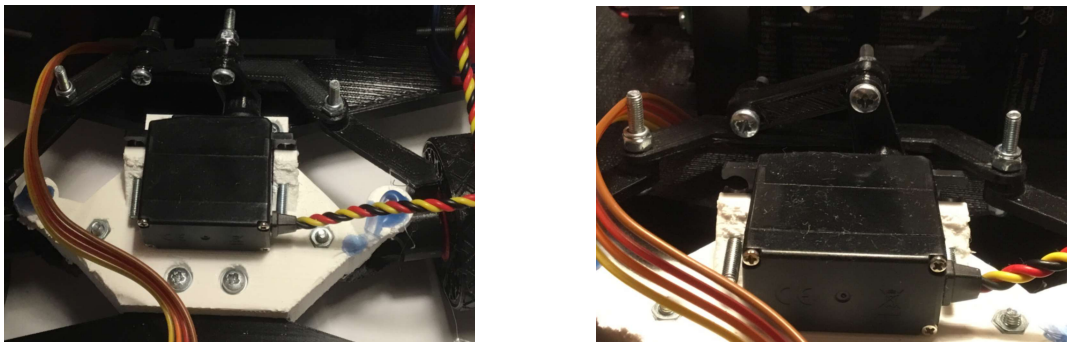


Figure 12: Illustration of how to attach the front wheel to the steering frame.

In the next step we want to connect the two wheels with its servo. When all the pieces are attached the system will look similar to figure 13a. The servo should be located in the center of the steering frame. Attach the servo horn at the pin at the backside of the servo.



(a) (b)

Figure 13: a) System for steering the two front wheels. b) Zoomed in on the link and steering link.

In the next step we will use the the link system build in section 3.3, which can be seen in figure 6. Connect the servo horn and the link with a screw (4.3x21), a hollow ball and a screw nut. The hollow ball should be located between the servo horn and the link. As the final step we want to connect to the steering link and the two wheel holders up left and right. Face the screw head downward as in figure 13b and fixate its position with two screws (4.3x21) and nuts.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSfz9naMdsIZgUkhrbUMYIZDeRN6F0wzwQhxG2mfpkZw9inh`

3.6 Attachment chips

Next we are going to assemble the chips and their attachment surfaces. One needs

- 1 Raspberry Pi holder

- 1 Raspberry Pi
- 1 sd-card
- 1 Beaglebone holder
- 1 Beaglebone
- 12 screws (4.3x21)
- 12 screw nuts

Before fixating the chips we will insert a sd card into the Raspberry Pi. The location and orientation of the card can be seen in figure 14. The orientation is essential and will be tricky to change later on.

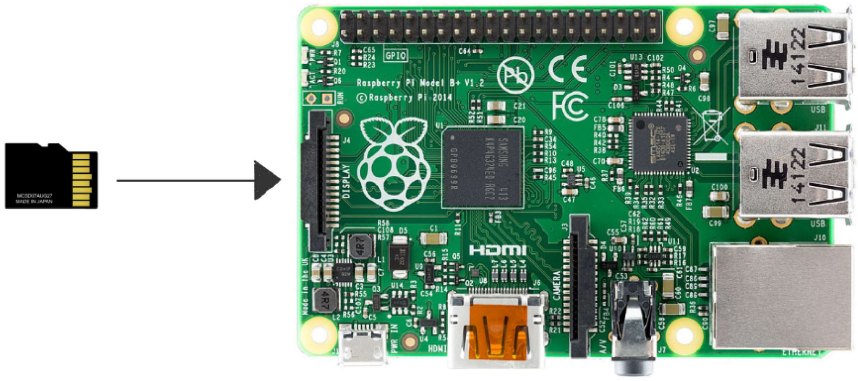


Figure 14: Location and orientation of SD card. The picture of the Raspberry and the SD card is under creative commons license at Wikipedia. Changes in the picture have been made by adding one arrows and two lines of text. Link to the picture of Raspberry Pi `https://commons.wikimedia.org/wiki/File:Raspberry_Pi_B%2B_top.jpg` . Link to the picture of SD card `https://upload.wikimedia.org/wikipedia/commons/6/67/SD_Cards.svg`

Next fixate the Raspberry Pi and the Beaglebone to their respective holder. For each chip one needs four screws (4.3x21) and screw nuts. Going back to the figure 8 illustrating the ground plate. We have two rectangular shape gaps, one on either side of the gap for the battery. Staring with the right one, where the attachment surface for the Beaglebone should be placed. Place the Beaglebone holder so its screw holes match the two holes next to the gap in the ground plate. Fix the Beaglebone holder with 2 screws and screw nuts. Repeat the same process for the Raspberry Pi holder.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSflEhELO6vEJeb3dhS8hu-Bt97Mpfn_oBUpn6bD-NrRTgQ`

3.7 Roof and doors

The roof and walls should be the very last step when building the Kiwi car. So return to this section when all the hardware and wires are connected. For this step one needs

- 2 side brackets
- 2 roof bridges
- 8 screws (4.3x21)
- 4 screw nuts

Figure 15 shows how the roof and sides will look like when all pieces are assembled.

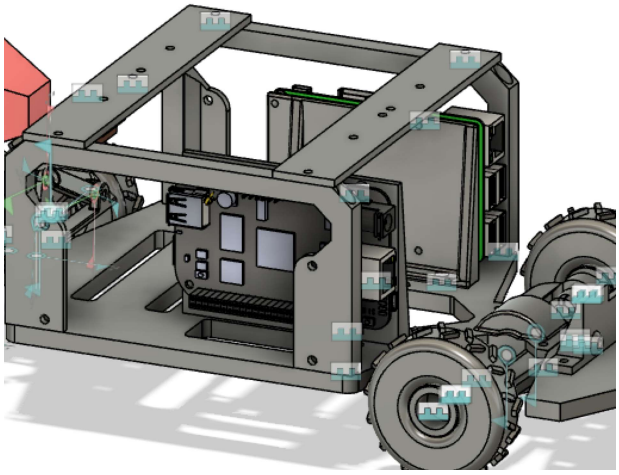


Figure 15: Illustration of the roof and the sides

Place both roof bridges at the top of the side brackets. One in the front and one in the back. Merge the sides and the roof with two screws for each roof bridge. Next step is to fixate the side brackets in the ground plate. It can be seen in figure 8 that the ground plate has one screw hole in each of the four corners. Match these holes with the screw holes in the side brackets and fixate with four screws (4.3x21) and the associated nuts.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSda04_M-_So8WkB_bz-Avk0f0F2JPsYh22_zokjvlPXJ41`

4 Technical equipment

In this section we will introduce the technical equipment. First focusing on where the equipment will be located and how we will integrate it with the chassis of the car and the surrounding technical components. Secondly technical information will be presented such as limitation, possible improvements and explanation how the equipment works. If you are only interested in building the car, the technical information is not crucial but provide a good insight into what the user can expect in terms of for instance the limitations of the sensors.

4.1 Daughter board

The daughter board collects most of the signals from the sensors, chips and so on. We are therefore going to start with building it and in the following section when integrating the technical equipment we are going to refer to this section for understanding how to connect them. But firstly, one should fixate the daughter board to the roof bridge, one needs

- 1 daughter board
- 1 roof bridge
- 2 screws (4.3x21)
- 4 screw nuts
- 4 screw washers

I would recommend to fixate the daughter board at the roof bridge before connecting the wires to increase the stability but there is nothing that impede the user to do it in the opposite order.

Face the screw head downwards, put on a screw washer before the roof bridge and a screw nut. Tighten the screw nut before attaching it on a screw washer followed by the daughter board and lastly a screw nut. Tighten the screw nut and repeat the process for the second screw hole in the roof bridge and the daughter board. When the daughter board is fixated on the roof bridge, it will look similar to figure 16.

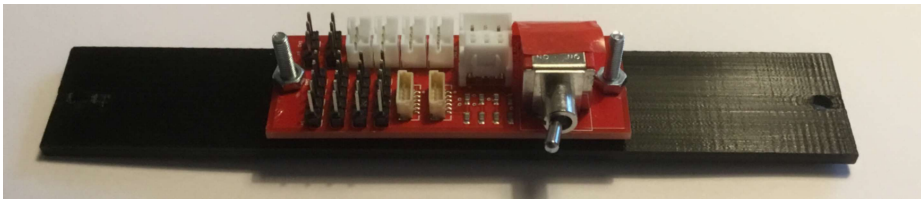


Figure 16: Daughter board fixated on the roof bridge.

In the end of this document when the chips, engine and battery is connected to the daughter board it will look like figure 17.

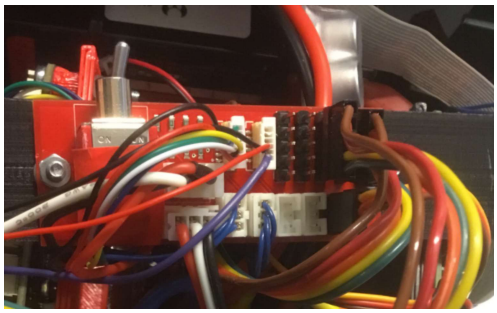


Figure 17: Daughter board with all connections

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLScXqoBGGbZYuRhlmnKPZ_t_oq7CT8OqttUTlqDaL9vCr6z8`

4.2 Battery and engine

Before working with the battery and engine make sure that you have read and understood the safety instructions in section 1.

For the battery and engine one needs

- 1 battery (7.4V lithium-ion polymer)
- 2 straps
- 2 lipo cable
- 1 crimp housing

In figure 8 there is two rectangular shape gaps in the front of the ground plate. Place the battery between these two gaps so that wires can be routed around the battery. It is important that the connectors at the battery are on the right side of the car. Use the straps to fixate the battery in the right position. It is beneficial to thread the straps between the two gaps and over the battery. Next we want to connect the battery to the daughter board by connecting the pd connector to the lipo cable, as can be seen in figure 18.

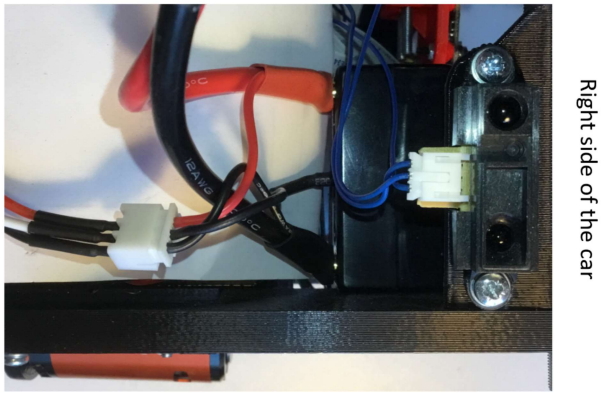


Figure 18: Illustration of how how to connect the lipo cable to the battery.

Connect the other end of the lipo cable to the daughter board in the same position as shown in figure 19.

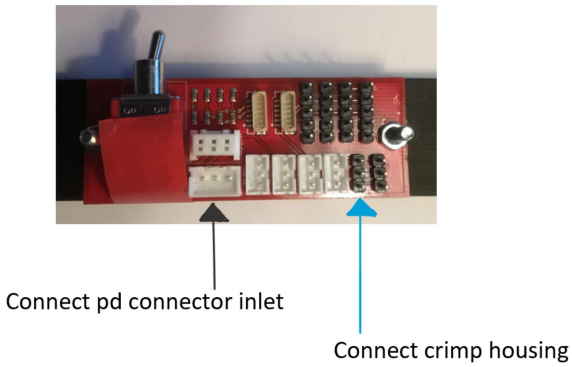


Figure 19: Connect the battery and the electronic speed controller to the daughter board.

We also need to connect the electronic speed controller (ESC) to the daughter board. As illustrated in the top of figure 11 there are three thinner wires (orange, red and brown) attached to the ESC. In the other end of the wires one needs a crimp housing. The crimp housing should be connected to the daughter board according to figure 19, with its flat black side facing left in the figure. The wires should be positioned orange, red brown, where the brown wire is closest to the blue arrow.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSd8ahmOyNPMOg49dd0QnBvwCHz3j7XBGHrzi4AJUTgAFhm`

4.3 BeagleBoard

The computational power of the car is divided between two chips, one Raspberry Pi and one Beagleboard. The Beagleboard is located at the left side of the car and was attach in section 3.6. To secure that the connection becomes correct we will use the information presented at the chip. In figure 20 one can see that the crimp housing sockets has numbered position as well as each one of the three sticks (-,+ ∘).

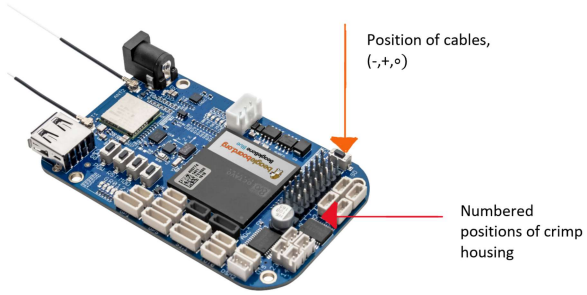


Figure 20: Numbered and symbolic position at the Beagleboard. The picture is owned by BeagleBoard.org Foundation.

In the end of this section, when all cables are attached to the Beagleboard the chip will have the same connections as illustrated in figure 21.

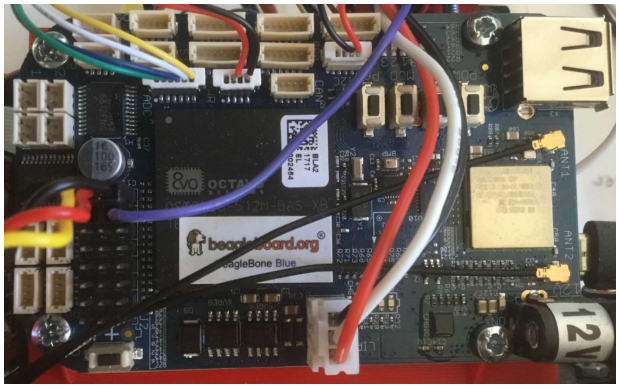


Figure 21: All connections to the Beagleboard.

Starting with the insurance of the power supply to the chip. We need

- 1 lipo cable

Connect the outlet of the lipo cable to the Beagleboard as illustrated in figure 22.

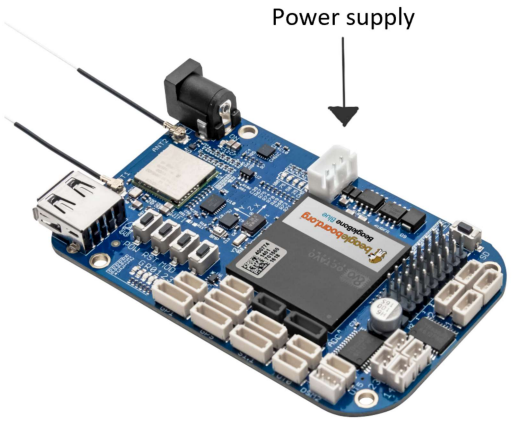


Figure 22: Power supply of BeagleBone. The picture is owned by BeagleBoard.org Foundation.

The other end of the lipo cable should be connected to the daughter board. How this is done can be seen in figure 23

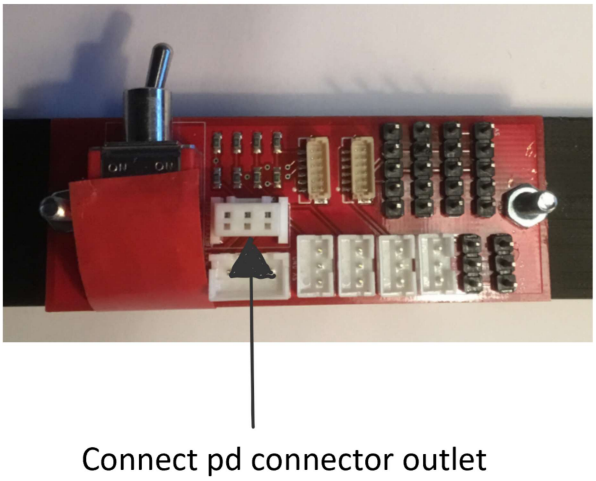


Figure 23: Connect daughter board to BeagleBoard power supply. The picture is owned by BeagleBoard.org Foundation.

In the next step we will connect the BeagleBoard to the servo. We need

- 1 servo

Connect the crimp housing attached to the servo in the position illustrated in figure 24. Going back to 20 the crimp housing should be placed in the first position and the yellow cable should correspond to the minus position, red to the plus position and the black to the circle position.

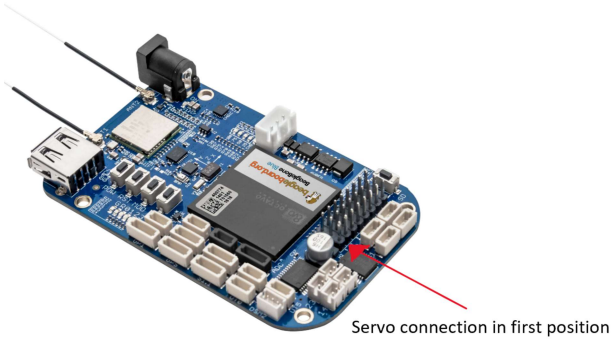


Figure 24: Connecting servo to BeagleBoard. Note that it should be positioned in the first position counting from the left side. The picture is owned by BeagleBoard.org Foundation.

Turning our attention to the jst and where to attach them on the Beagleboard chip. One needs

- 1 jst inlet (4.9mm long 4.8mm wide)
- 2 jst inlet (4.9mm long 6.8mm wide)
- 6 jst wires

Start with one jst inlet (4.9x4.8) and one jst inlet (4.9x6.8). The larger of the two should have four jst wires leaving two positions unused and the smaller one should have two wires, also leaving two positions unused. Connect the jst inlet to the black jst outlet in the BeagleBoard as shown in figure 25.

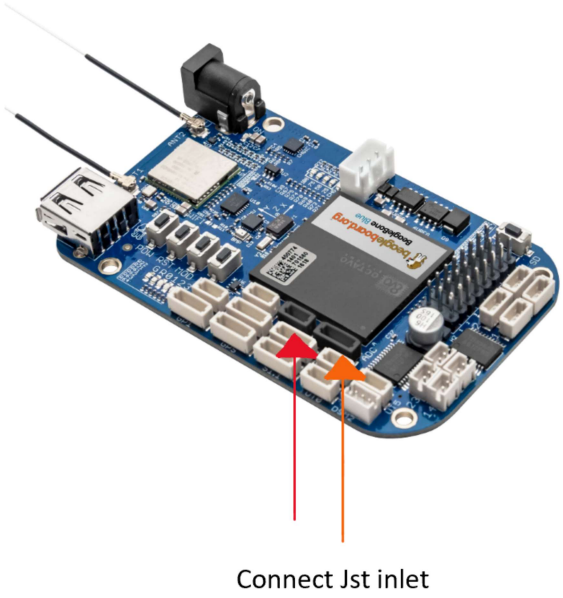


Figure 25: Illustration of where the jst inlet should be connected to the BeagleBoard. The picture is owned by BeagleBoard.org Foundation.

To later on be able to connect the BeagleBoard to the daughter board we needs to attach the six free wires to a jst inlet (4.9x6.8). In which order the wires are positioned are of great importance so follow the instructions in figure carefully 26. Notice that the colour of the wire is insignificant, it is only its position in the jsts that is critical.

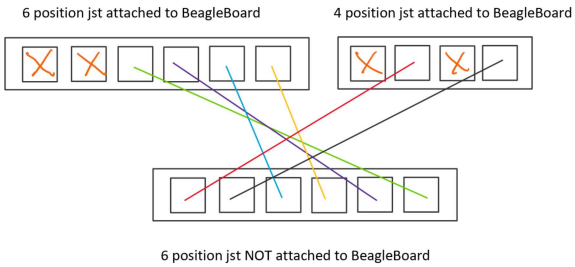


Figure 26: Illustration of how to connect the jst wires. Notice that the colour of the wires are irrelevant. The two jsts at the top is connected to the BeagleBoard. The picture is owned by BeagleBoard.org Foundation.

Connect the new jst, the one that is not attach to the BeagleBoard, to the daughter board as shown in figure 27.

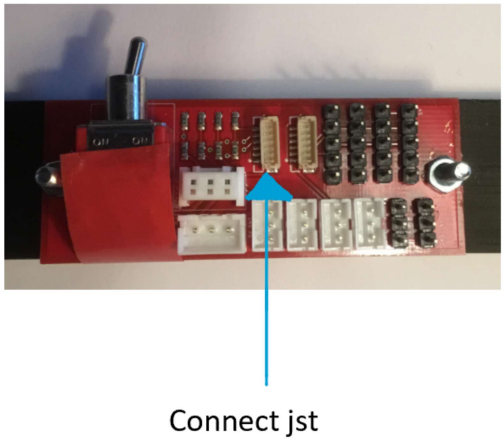


Figure 27: How to connect the jst with six wires to the daughter board.

Next one needs

- 1 jst inlet (4.9mm long 4.8mm wide)
- 1 jst inlet (4.9mm long 6.8mm wide)
- 1 crimp housing
- 2 jst wires
- 1 LiYv wire

We have one last connection between the BeagleBoard and the daughter board to do. The smaller jst (four positions) and the crimp housing should be attach to the BeagleBoard according to figure 30. The black flat side of the crimp housing should be faced downwards when attached to the Beagleboard. Notice the position of the crimp housing in the figure.

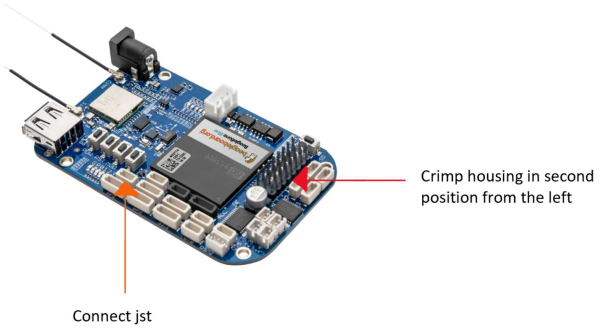


Figure 28: Illustration of how to connect the remaining jst and the crimp housing to the BeagleBoard. The picture is owned by BeagleBoard.org Foundation.

Connect the wires to the larger jst (6 position) according to figure 29. The red and black wires from the jst is jst wires, the blue thicker wire is a LiYv wire. Notice that the colour of the wires are irrelevant but how they are positioned are of great importance. Even though we only have three wires we still need the larger jst to connect it to the daughter board.

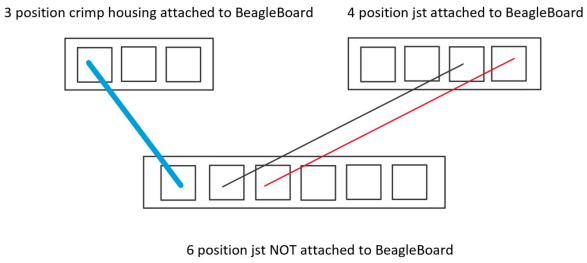


Figure 29: Where to connect the jst and crimp housing to the BeagleBoard. The picture is owned by BeagleBoard.org Foundation.

Connect the jst to the daughter board as shown in figure 30.

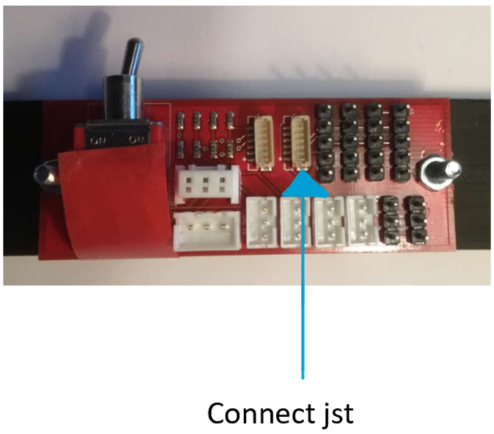


Figure 30: How to connect jst to the daughter board.

Using the Beagleboard we can also add wifi antennas to the car. These will be useful when driving our car since it will be interacting with an online interface. We need

- 2 Wifi antennas

The position of the wifi antennas is not crucial, but a suggested position is to the right of the gaps for the chips in figure 8. Attach the other end of the antenna to to Beagleboard as illustrated in figure 31.

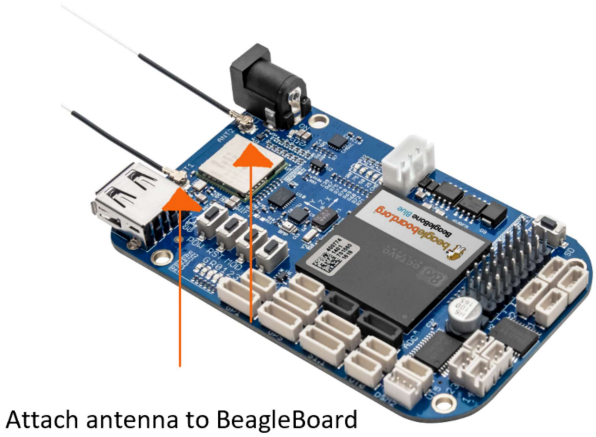


Figure 31: Where to attach antenna to Beagleboard. The picture is owned by BeagleBoard.org Foundation.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSeaIKNqWpMA2QAquO-Qig2sl4uzK6CzV5qaaasPsHEaBYr`

4.4 Raspberry Pi

Before connecting the Raspeberry Pi chip we need to attach the camera to the front roof bridge. One needs

- 1 roof bridge
- 1 Raspberry Pi camera
- 2 screws (4.3x21)
- 2 screw nuts (inner diameter 2.7mm)
- 1 picam holder
- 4 screws(3.6x11.3)
- 4 screw nuts (inner diameter 1.7mm)
- 1 usb cable
- 2 crimp housing
- 2 LiYv wires

When assembling all the pieces the camera will look similar to figure 32. Fixate the Raspberry Pi camera at the Picam holder with four screws (3.6x11.3) and nuts, one in each corner. To simply, let the tape at the camera be faced downwards. Next attach the Picam holder to the roof bridge with two screws (4.3x21) and nuts.

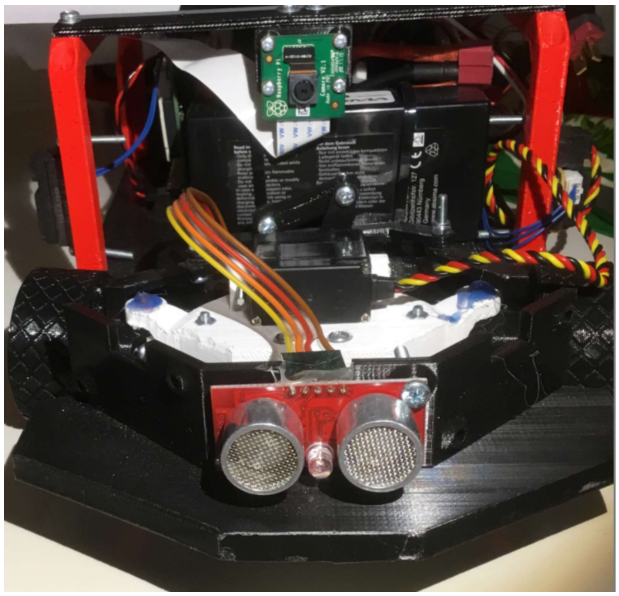


Figure 32: Illustration of the camera attached to the roof bridge.

Next we will focus on the Raspberry Pi. When all connections are made the chip will lock similar to figure 33.

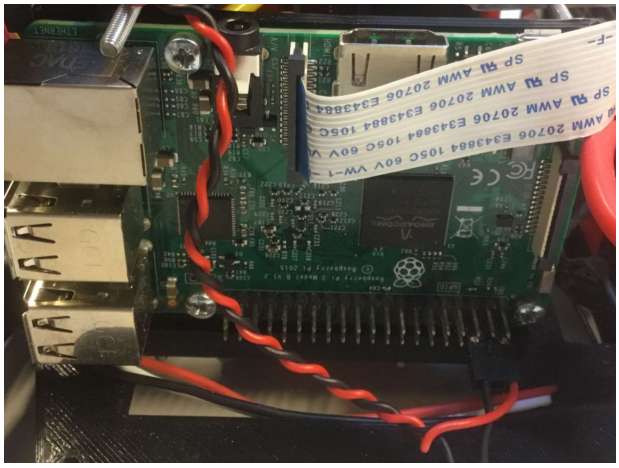


Figure 33: Illustration of the Raspberry Pi with all connections.

Attach the other end of the camera tape to the Raspberry Pi chip according to figure 34. To power the chip we need to connect it with an usb cable to the BeagleBoard chip. Where figure 34 indicates one should connect the usb cable there is two ports, choose the port closest to the chip. Connect the other end of the cable to the BeagleBoard as illustrated in figure 35. It is now suitable to return to section 3.7 and attach the two side brackets and the front roof bridge to the car, but wait with the back roof bridge.

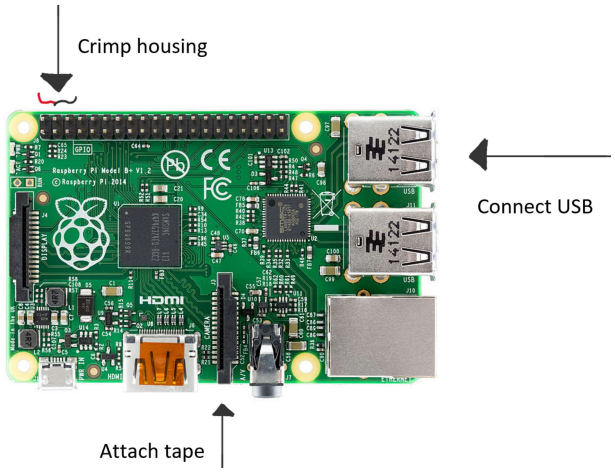


Figure 34: Connections at the Raspberry Pi chip. The picture is under creative commons license at Wikipedia. Changes in the picture have been made by adding two arrows and two lines of text. Link to the picture `https://commons.wikimedia.org/wiki/File:Raspberry_Pi_B%2B_top.jpg`.

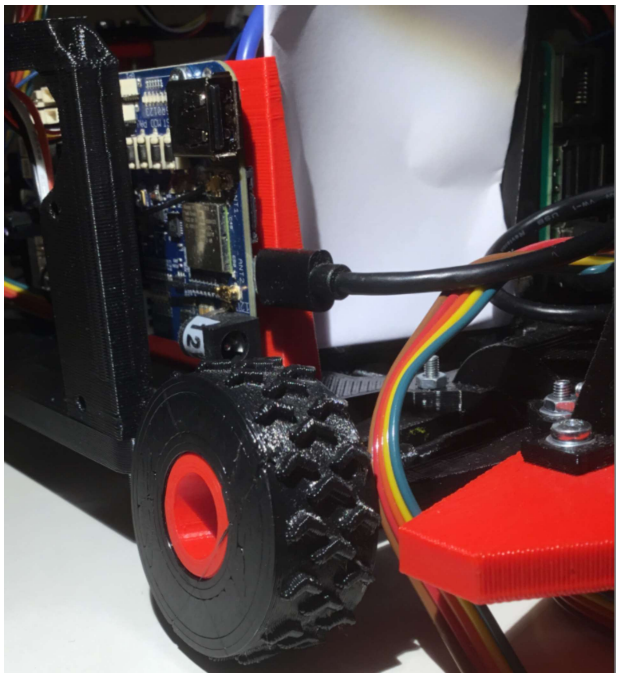


Figure 35: How to connect usb cable to BeagleBoard.

For the last connection we need two crimp housings. Facing the flat black side downwards the first wire should be connected in the third position count from the left in the first crimp housing and in the second position in the second crimp housing. The second wire should be in position one in the first crimp housing and in position three in the second crimp housing. This is illustrated in figure 36.

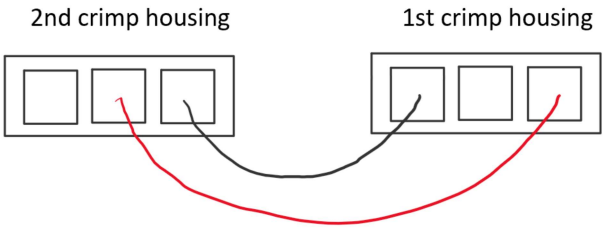


Figure 36: How the wires between the two crimp housings should be connected

Connect the first crimp housing to the Raspberry Pi chip as illustrated in figure 34. Following the colours at the wires in figure 36 the red wire should be connected to the red side of the brackets in figure 34. Connect the second crimp housing to the daughter board as shown in figure 37. Make sure that the flat black side of the crimp housing is faced in the direction of the Raspberry Pi chip.

Figure 37: Where to connect crimp housing from Raspberry Pi to daughter board.

It is now suitable to return to section 3.7 and attach the two side brackets and the front roof bridge to the car, but wait with the back roof bridge.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSdKytBwUU1r6oLwUPhgOU3wAPcu_5PVvcMG_VfigceujwR`

4.4.1 Purpose of the chips *

The version of the Raspberry Pi used in the car is The Raspberry Pi 3 Model B+ which will remain in production until at least January 2023. If using a newer version the specification and link below may not be up to date. The processor has 64 bits and is much more powerful than the Beaglebone. Therefore the Raspberry Pi chips is responsible for the heavy calculations such as image processing. For more specifications visit `http://www.farnell.com/datasheets/2911643.pdf`.

The Beaglebone chip is a low level computer board and is responsible for communicating with the sensors. With the features available on Kiwi car the chip is sparingly used and several new features can be added to the car for those who are interested. For examples on features and specifications check out `http://www.farnell.com/datasheets/2237500.pdf`. In the upper right corner of the chip, when it is attach to the car, there is four buttons. The third one (RST), counting from left to right is a reset button and the forth (POW) is the power button.

4.5 Distance sensors

There is four distance sensors on the car. Two Ultrasonic sensors, one in the front and one in the back and two IR sensors at the front side brackets. The location can be seen in figure 38.

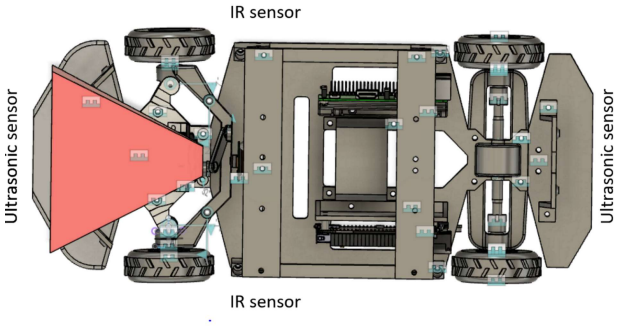


Figure 38: The Kiwi car seen from above, with the location of the sensors.

4.5.1 Ultrasonic sensors

Starting with the front sensor we need

- 1 Ultrasonic Range Finder - I2C
- 1 sensor holder front
- 4 screws (4.3x21)
- 4 screw nuts

After attaching the Ultrasonic sensor the front bumper will look similar to figure 39.

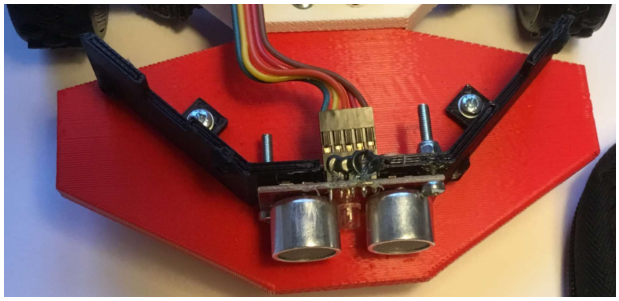


Figure 39: Front sensor

Fix the Ultrasonic sensor at the flat side of the sensor holder facing forward, using two screws (4.3x21) and nuts. Position the sensor holder so its screw holes match the ones in the front bumper and fixate them. For the back sensor one needs

- 1 Ultrasonic Range Finder - I2C
- 1 sensor holder back
- 4 screws (4.3x21)
- 4 screw nuts

When attaching the Ultrasonic sensor to the back bumper, it will look similar to figure 40. Repeat the steps for the front sensor by fixate the sensor at the sensor holder back. One will need two screws (4.3x21) and two nuts. Match the screw holes in the sensor holder with the ones in the back bumper and fixate them.

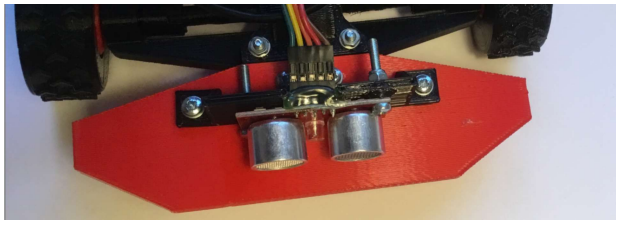


Figure 40: Back sensor

4.5.2 Connect sensors

Starting with the Ultrasonic Range one needs

- 2 ribbon cables
- 4 crimp housing

Attach the crimp housing with the ribbon cable to the front and back sensor as can be seen in figure 39 and 40. To connect the sensor to the daughter board we also need a crimp housing in the other end of the ribbon cable. Make sure that the black flat side of the crimp housing is facing downwards in both edges of the ribbon cable. Connect the crimp housing to the daughter board as shown in figure 41.

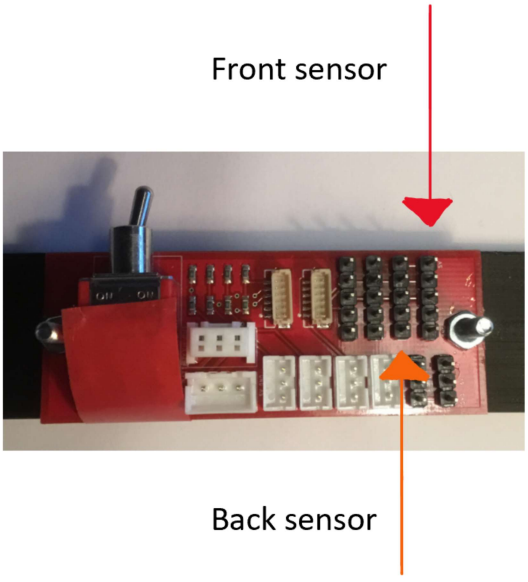


Figure 41: How to connect the Ultrasonic sensors to the daughter board.

4.5.3 IR sensors

For the IR sensors one needs

- 2 IR sensors
- 2 screws (4.3x21)

- 2 screw nuts
- 3 jst wires (where each end is connected to a jst in or outlet)

At the side bracket there is two enlargement at the left and right side of the bracket. Place the IR sensor so its screw holes match the ones in the enlargement of the side bracket. To avoid unnecessarily tangled wires, let the white jst outlet attached to the sensor point in the direction of the BeagleBoard chip. Fixate the sensor with two screws and nuts. It may be useful to temporary remove the front roof bridge to get a better grip.
Attach a jst inlet with three wires to the IR sensor. Most likely the jst wires are preattached to the jst in and outlet. If not, make sure that the positions of the wires in the in and outlet corresponds. Connect the jst to the daughter board as shown in figure 42, where the right IR sensors refers to the right side of the car. Repeat the steps for the two IR sensors.

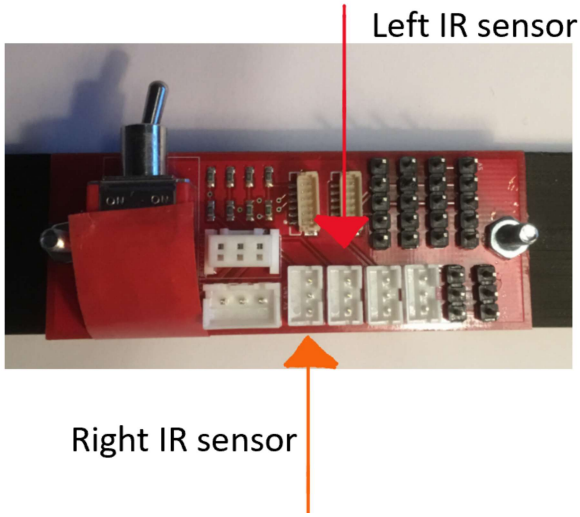


Figure 42: How to connect the IR sensors to the daughter board.

All the connections to the daughter board is now finished. Go back to section 3.7 and attach the back roof bridge.

4.5.4 Performance limitations Ultrasonic sensors ∗

The sensor on the left and the right side of the car are infra red sensors. To measure the distance as precisely as possible, avoid exposing the sensor to other light sources. If this is unavoidable one may use some kind of protection. Choose a protection that is an efficient transmittance at the emitting wavelength range of LED ($\sim$ λ870nm $\pm$70nm). Furthermore, the IR sensor provides the best result at smooth surfaces and the lens must be held clean from dust, water and oil ect. The sensor translates the voltage of the incoming signal to a distance and is capable of measuring from 4 to 30 cm (1.57-11.81 inches). When reaching a distance of 30cm to the obstacle ahead the voltage of the signal will drop to zero, meaning that the same value will be given for all distances longer than 30cm.

Ultrasonic Range Finder is capable of measuring up to seventeen object in a distance from 3cm up 6m. This is done with 40kHz pulses. The maximum of seventeen objects that are detected are sorted after shortest distance to the car and the distance to the object is approximated with an interpolation.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSfRnvvl2jI-sKWYESujEQM3HVEXPMcc_1_XhJ2SLBMZrA7`

5 Start up the Kiwi car

This section will provide information for how to start up the kiwi car and connect to its wifi.

5.1 Booting the system

There is a specific sequences to follow in order to boot up the Kiwi platform correctly. First, connect the main connectors as can be seen in figure 1. Secondly turn on the Beaglebone by using the main power switch located at the daughter board. You will see that there is a red lights blinking at the front and back Ultrasonic sensors for a few seconds followed by several blue lights blinking at the Beaglebone. Wait until the green or if the battery is low, red lights in the lower part of the Beaglebone are turned on. The locations of the led lights can be seen in figure 43. Now you can turn on the ESC by pushing its power button. The Kiwi car should now begin to beeping repeatedly and a solid red light combined with a blinking green lights should be seen at the upper right corner of the Raspberry Pi chip. The beeping sound indicates that the beaglebone is waiting to connect to the Raspberry Pi. When you see a red led light in the top of the right corner of the beaglebone, push the following buttons in sequence, MOD, PAU, MOD. If this is done correctly the beeping sound should have stop and the red light should have turned green.

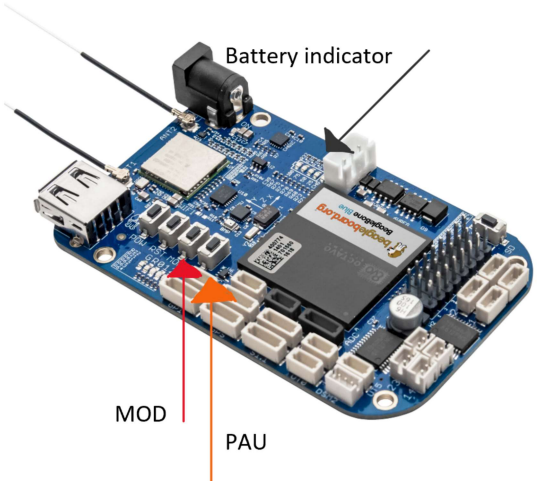


Figure 43: Beaglebone marked with MOD and PAU buttons and the battery indicator. The picture is owned by BeagleBoard.org Foundation.

A demonstration how to start the car can be seen in this link: `https://youtu.be/SIavbNt_MUM`.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSdlzZ3h5lJ5sql0UQzGUt41ucl9HE4vAOjwsKt7ILK0hwhs`

5.2 Connect to Wifi

Kiwi comes with Wifi enabled by default and is configured to share an Internet connection. Search in your Wifi settings for the access point (AP). Most likely the Wifi will be named similar to BeagleBone-11XX and the password is *BeagleBone*. Your laptop should receive an IP adress similar within the range 192.168.8.xxx. You can now ping your computer to the Kiwi or connect via ssh using (user: debian, password:temppwd) for the beaglebone and (user:pi password:raspberry) for the raspberry pi. The commands in the terminal is *ssh debian@192.168.8.1* for the beaglebone and *ssh -p 2200 pi@192.168.8.1* for the raspberry pi.

A demonstration how to start the car can be seen in this link: `https://youtu.be/Dfbn-5dmJm4`.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSf1pcGpC72dY9dcpoAKKCMOdqO71SWSe2YAp6wipCFBxQl8`

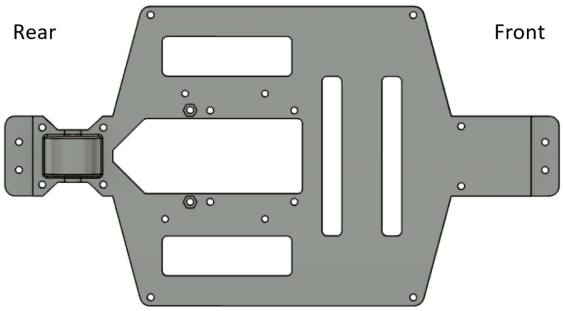


Figure 44: Base plate

Figure 45: Wheel

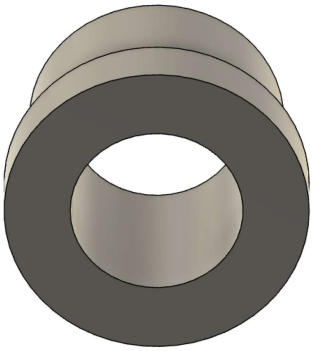


Figure 46: Distance

Figure 47: Rim

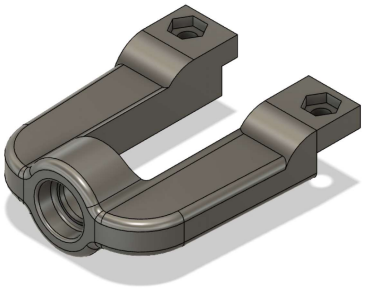


Figure 48: Tyres brackets

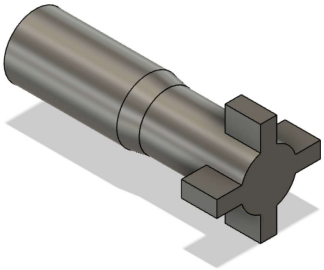


Figure 49: Wheel shaft

Figure 50: Cup joint

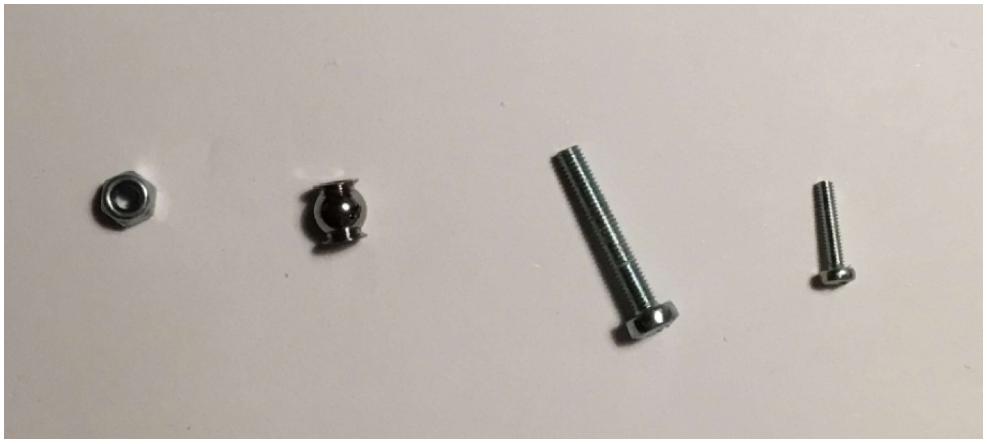


Figure 51: From left to right, screw nut, hollow ball, screw (4.3x21), screw (2.3x13)

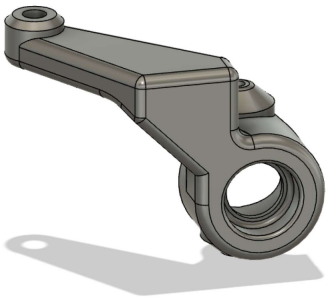


Figure 52: Wheel holder up right. Wheel holder up left is an mirror image of the right one.

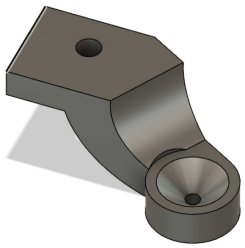


Figure 53: Wheel holder down right. Wheel holder down left is an mirror image of the right one.

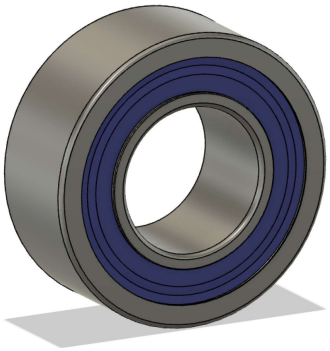


Figure 54: Bearing

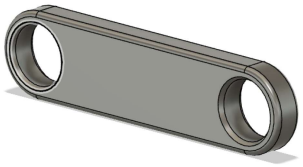


Figure 55: Link

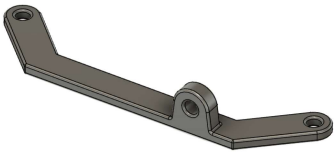


Figure 56: Steering link

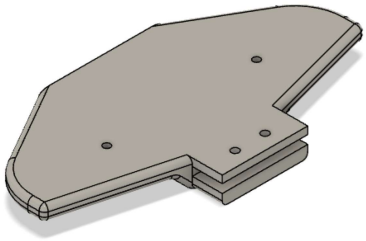


Figure 57: Front bumper

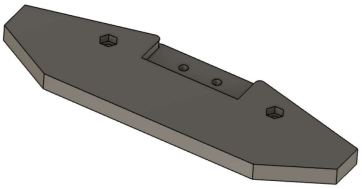


Figure 58: Rear bumper

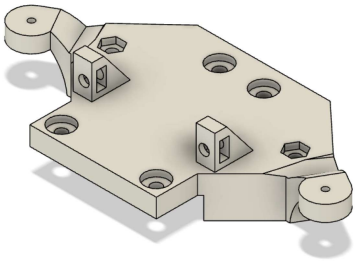


Figure 59: Steering frame

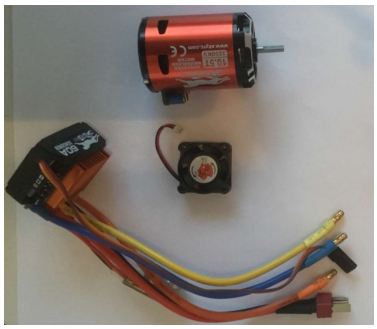


Figure 60: ESC and battery

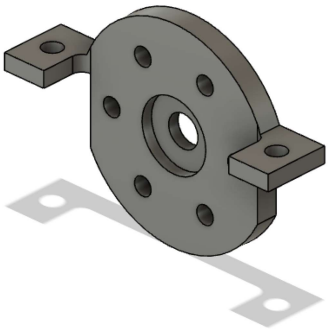


Figure 61: Motor holder

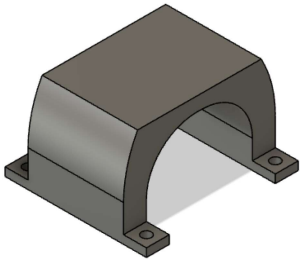


Figure 62: Motor housing

Figure 63: Shaft connector

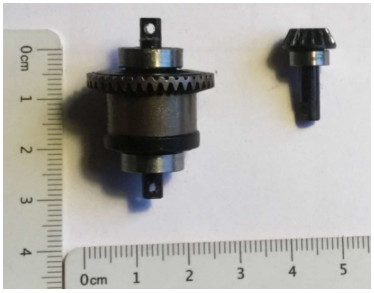


Figure 64: Gearbox at the left and transfer pin to the right.

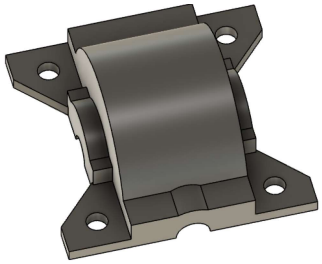


Figure 65: Differential housing

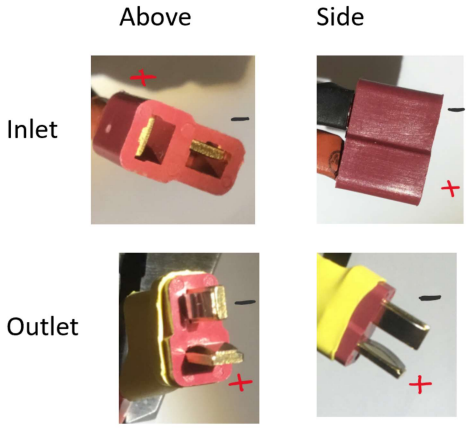


Figure 66: Battery base T-connector system. The red plus indicates the plus pole and the black minus, the minus pole.

Figure 67: Servo

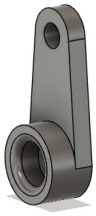


Figure 68: Servo horn

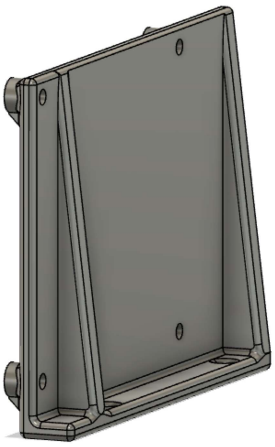


Figure 69: Raspberry Pi holder

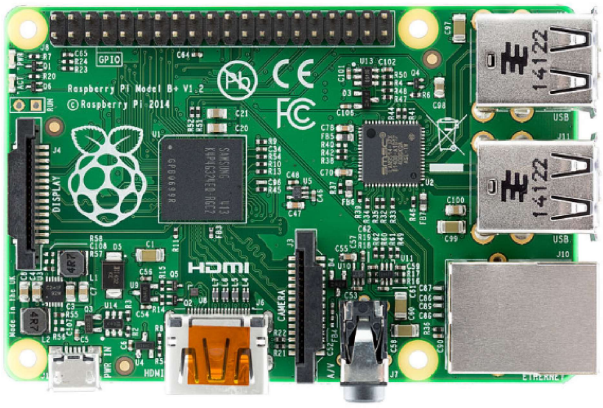


Figure 70: Raspberry Pi chip. The picture is under creative commons license at Wikipedia. Changes in the picture have been made by adding two arrows and two lines of text. Link to the picture `https://commons.wikimedia.org/wiki/File:Raspberry_Pi_B%2B_top.jpg`.

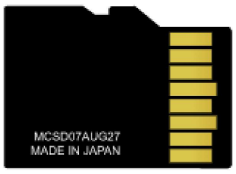


Figure 71: Sd card. The picture is under creative commons license at Wikipedia. Link to the picture `https://upload.wikimedia.org/wikipedia/commons/6/67/SD_Cards.svg`

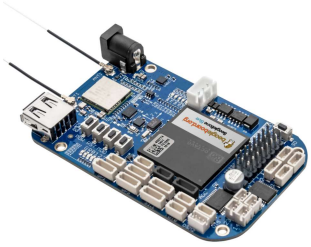


Figure 72: Beaglebone chip. The picture is owned by BeagleBoard.org Foundation.

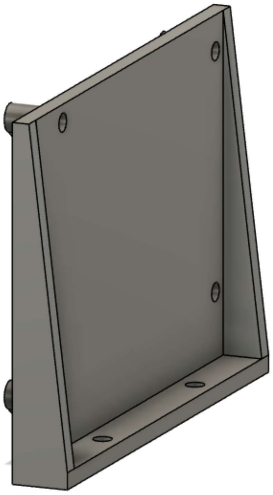


Figure 73: Beaglebone Holder

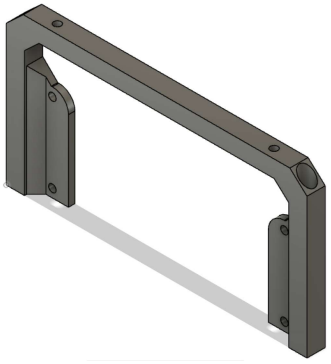


Figure 74: Side brackets

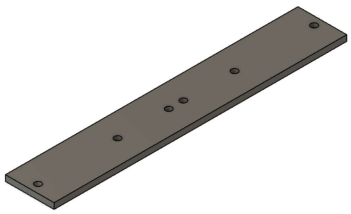


Figure 75: Roof bridge

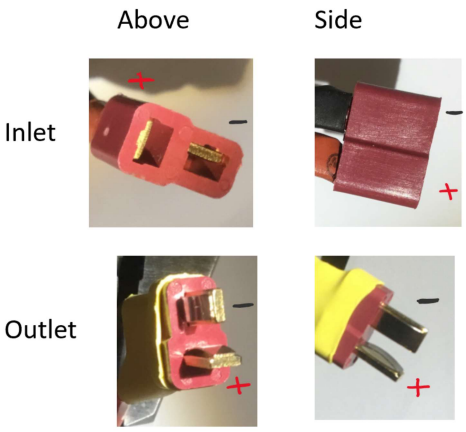
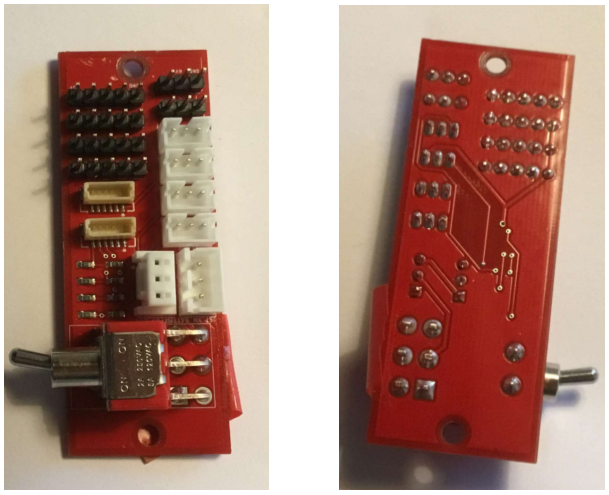


Figure 76: Battery base T-connector system. The red plus indicates where to connect the plus pole and the black minus where to connect the minus pole.



(a) (b)

Figure 77: a) Seen from above b) Seen from below.

Figure 78: Battery wire

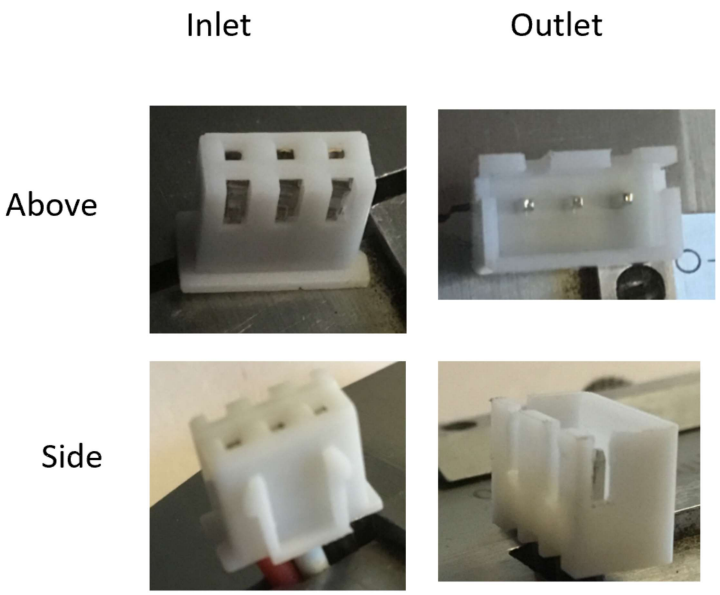


Figure 79: In and outlet for pd connectors seen from above and aside

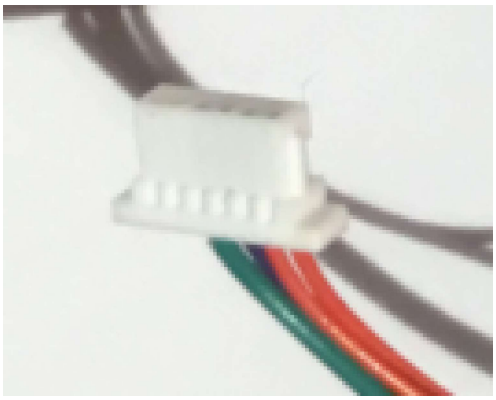


Figure 80: Jst inlet (4.9mm long 6.8mm wide). The smaller one looks similar.

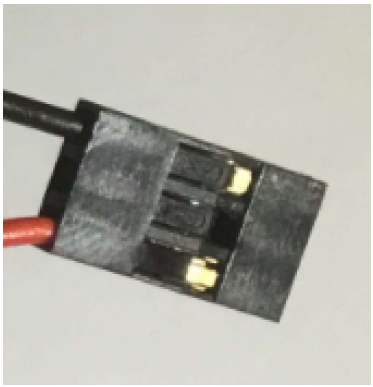


Figure 81: Crimp housing

Figure 82: LiYv wire. Note that the jst wire is looking the same but has a smaller inner diameter.

Figure 83: Lipo cable

Figure 84: Wifi

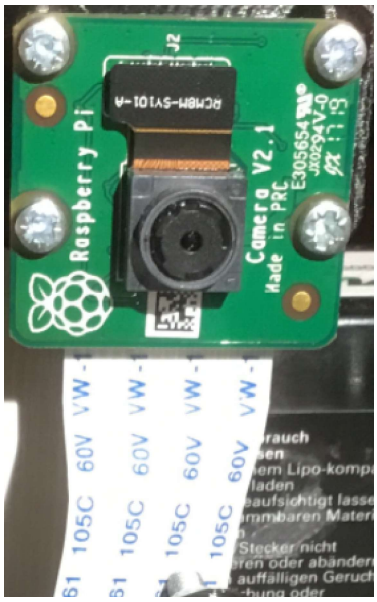


Figure 85: Raspberry Pi camera

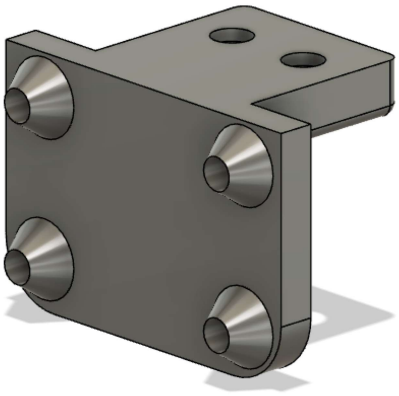


Figure 86: Raspberry Pi camera holder

Figure 87: Usb cable

Figure 88: Ultrasonic sensor

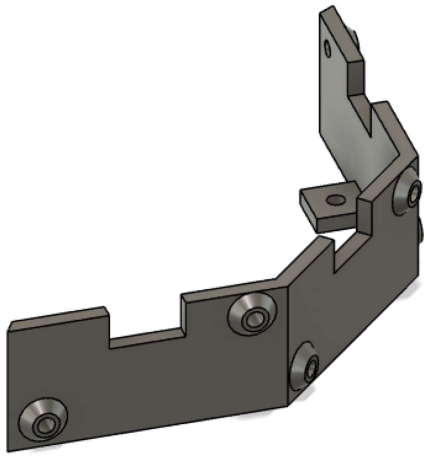


Figure 89: Sensor holder front

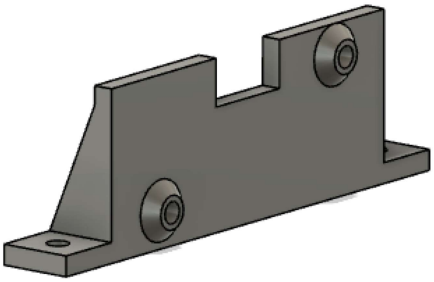


Figure 90: Sensor holder back

Figure 91: Ribbon cable

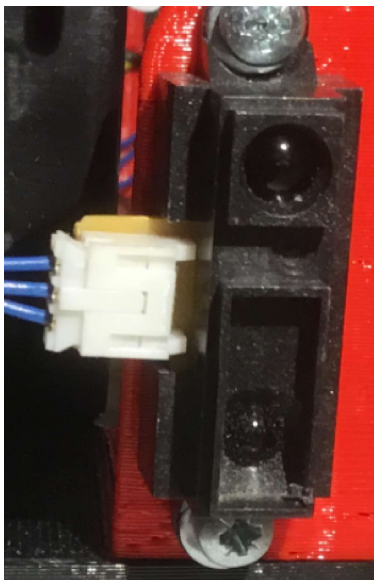


Figure 92: IR sensor

E

Tutorials

1 Setting up the environment

Through this document we will mainly work with three tools, the Kiwi platform, our PC and Gitlab. A first glance of how they are linked together can be seen in figure 1.

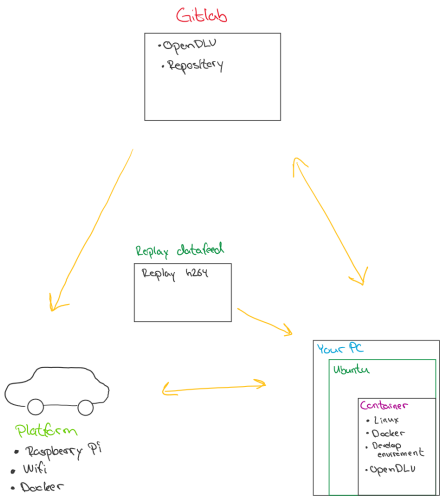


Figure 1: Simplified model of tools in this document.

This document will assume that you are working on a Linux operative system. If you are running on a Windows or a Mac you can create an Virtualbox in this link. Next create a Ubuntu, a Linux operating system by pressing new. All the steps can also be followed in this link. Make sure that you have some extra memory in the last step, *File location and size*. Increasing the memory will not be possible afterwards. Next the image file for Ubuntu must be installed, you can found the website here. Download the file and make sure that you have the same version as the created virtual machine. In the tutorial the version being used supports 64 bits. The first time you run Ubuntu it will ask for the startup disk. Locate the file you just downloaded and add and select it. Ubuntu will now start booting up. The remaining step is to install Ubuntu and set your personal settings.

To be able to follow the upcoming tutorials we need to install some programs and packages. If you have previously worked with Ubuntu you may already have installed the programs and packages and do not need to do it again.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install build-essential
```

- *sudo apt-get update* checks for software updates for Ubuntu
- *sudo apt-get upgrade* updates if new software is available
- *sudo apt-get install build-essential* is a package of software for Ubuntu containing compilers for example.

In these tutorials we will write smaller software projects in Python. If we would write larger projects in C++ for example a CMake file would be beneficial as a script to help us execute the right steps in order.

```
sudo apt-get install cmake
```

- *sudo apt-get install cmake* installs cmake

Later on, in the tutorials we will work with UDP multicast. This means that we can send small volume of data which can be reach by all the computers in the same network.

```
sudo apt-get install socat
```

- *sudo apt-get install socat* installs cmake

To be able to communicate between the processes running on the car we need to set up a protocol for the messages. How the messages work will be explained in section 5.1.

```
sudo add-apt-repository -y ppa:chrberger/libcluon
sudo apt-get update
sudo apt-get install libcluon
```

- *sudo add-apt-repository -y ppa:chrberger/libcluon* adds the repository where messages software package libcluon is located
- *sudo apt-get update* checks for software updates
- *sudo apt-get install libcluon* installs libcluon

In the following tutorials we will use Gitlab to store our code but also to push it to the Kiwi car. Start with creating your own repository. To push our code from the computer to Gitlab we will use the command git.

```
sudo apt-get install git
```

- *sudo apt-get install git* installs git

The command git clone can be used to clone a repository. Go to this link. How to clone a repository can be seen in this link. Note that you need the specific addresses for each repository.

To build and run our code we will use Docker, which is further explained in section 5.4.

```
sudo apt-get install curl
-fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb
_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce
sudo usermod -aG docker $USER
```

The commands for installing docker is copied from Dockers website. If you are getting error you can copy the command from website. Here you can also find more information about the command. After completed the steps Ubuntu needs to be restarted.

Try to run the command *docker run hello-world*. If docker can download an image the installation is completed and the hello-world program will execute.

The last program we will install is docker-compose. It is useful if we want to run many processes at the same time but avoid starting each of them with docker run.

```
sudo curl -L https://github.com/docker/compose/releases/download/1.20.1/docker-compose-
‘uname -s‘-‘uname -m‘ -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

The commands for installing docker-compose is copied from Dockers website. If you are getting error you can copy the command from website. Here you can also find more information about the command.

To verify that the installation was successful, type *docker-compose --version*. The output should be the current version of docker-compose.

To install Python packages we use *sudo apt-get install --no-install-recommends* followed by the packages. The recommended packages for python3 are

```
build-essential
python3-protobuf
python3-sysv-ipc
python3-numpy
python3-opencv
protobuf-compiler
```

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLScOX0JTT2O9m-wVLKAjC5RpMt1Z-lTa1X9wzyJd`

2 Booting the system

The components mentioned in this part is identical with the ones being used in the manual for building the Kiwi car. Start with connecting the main connectors. Secondly turn the main power switch located at the daughter board and press the power button at the cheetah sensor positioned in the center of the car. The car will now start beeping and a blue flashing light can be seen at the Beaglebone-board. When one can see a red led light in the top of the right corner of the beaglebone, push the following buttons in sequence, MOD, PAU and MOD. If this is done correctly the beeping sound should have stopped and the red light should have turned green.

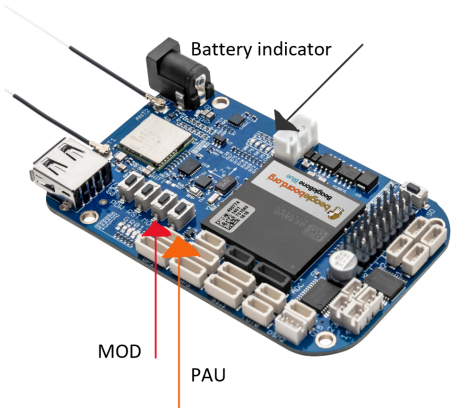


Figure 2: Beaglebone marked with MOD and PAU buttons and the battery indicator. The picture is owned by BeagleBoard.org Foundation.

A demonstration of how to start the car can be seen in this link.

If the battery is low and needs to be recharged it is important that the switch located at the daughter board is facing in the same direction as when the system is booting up. Otherwise the battery wont charge.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSdSYYV3iCBT0tFMB-dj8Ot75xEaCWeruzGRwePR`

2.1 Sources of error

If the system doesn't boot correctly, try to restart the system. If the error is still present, connect to Raspberry Pi to a screen by a HDMI cable. All the startup sequences will then be shown and the error can more easily be found.

3 Connect to Wifi

Kiwi comes with Wifi enabled by default and is configured to share an Internet connection. Search in your Wifi settings for the access point (AP). Most likely the Wifi will be named similar to *BeagleBone-11XX* and the password is *BeagleBone*. Your laptop should receive an IP address similar within the range 192.168.8.xxx. You can now ping your computer to the Kiwi or connect via ssh command using (user: debian, password:temppwd) for the Beaglebone and (user:pi password:raspberry) for the raspberry pi. The commands in the terminal is *ssh debian@192.168.8.1* for the Beaglebone and *ssh -p 2200 pi@192.168.8.1* for the raspberry pi.

A demonstration of how to start the car can be seen in this link.

When you are connected to the raspberry Pi follow this link to get access to your home network to the raspberry Pi.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSfDCzn7nbYjGtBRgq_2RmLaG8RXfVc7Vm6xODf7`

3.1 Sources of error

Nothing happens when I run the ssh command

1. Are you connected to the Beagleboard network?
2. Have you double checked the command?

3. Troubleshot the Raspberry pi, it may not have booted correctly (see section 2)

I can only ssh to debian

1. Troubleshoot the Raspberry Pi, it has most likely not booted correctly

The passwords does not work

1. The default password for the Beagleboard is temppwd and for the Raspberry Pi, raspberry. The password can be changed after accessing the first time.

4 Control the robot

Make sure that you are connected to the Beagleboard wifi and type 192.168.8.1:8081 in your web browser. You will identify an interface where you can see the sensors to the right and camera feed to the left. In the top left corner, you will receive a button called Messages. Here you can see which messages that are sent and received as well as the data and timestamp. In the center of the screen there are four buttons. By pressing the second from the left you can record the collected data. To download it, press the third button from the left and choose which file you want to download.

A demonstration of the interface can be seen in this link.

One can also steer the car manually. By pressing the button illustrating a joystick, a new tab is opening. The robot can then be steered with the mouse pad.

An illustration of how to steer the robot can be seen in this link.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSfeViA8-RRNuwQjCdsgrFZViOZGHZyG7ugxo_Cwt`

4.1 Sources of error

The camera does not work or is very slow

1. The tape between the camera and the Raspberry Pi may be damaged or be loose.

I can't access the website

1. The website is accessed by writing 192.168.8.1:8081 in the browser where 192.168.8.1 is the same address when connecting to the Raspberry Pi by ssh command. One can also write localhost:8081.

5 Writing the first code

We are now going to start producing our Python code to control the car. In section 1 we cloned a repository with the file myApplication.py. to be able to compile the code correctly, write your code in that file. In this link one can find very useful code examples and the commands that we will work with in these tutorials.

5.1 Messengers

To communicate with the sensors at the Kiwi car we use opendlv standard messages which can be found in the link. An example can be seen in figure 3 the message is of type opendlv.proxy.*specificationOfMessage* with an unique message ID(id = 1030). The data is stored in float variables also with an unique ID (accelerationX [id = 1]).

```
message opendlv.proxy.AccelerationReading [id = 1030] {
  float accelerationX [id = 1];
  float accelerationY [id = 2];
  float accelerationZ [id = 3];
}
```

Figure 3: Example of an acceleration reading message.

Other types of messages that can be useful are for example

opendlv.proxy.GroundSteeringRequest
opendlv.proxy.GroundSteeringReading
opendlv.proxy.PedalPositionRequest
opendlv.proxy.PedalPositionReading
opendlv.proxy.ImageReading

5.2 Receiving messages

To be able to handle messages we must set up a session for receiving and sending messages, which is done with an *OD4Session*. What the OD4Session doing is that it creates a communicate channel for listening and sending messages, in this case from the Kiwi car.

```
session = OD4Session.OD4Session(cid=112) #112 is cid for kiwi car
```

To update the distances we create the function onDistance. The sender stamp is predefined and the front sensor has senderStamp == 0 and the rear sensor senderStamp == 1. msg.distance is used to extract the message since the instance variable of opendlv.proxy.DistanceReading is called distance. For clarification check the opendlv-reference of standard messages in section 5.1.

```
distances = {"front":0.0, "rear":0.0};


def onDistance(msg, senderStamp, timeStamps):

        if senderStamp == 0:
                distance["front"]= msg.distance

        if senderStamp == 1:
                distance["rear"] = msg.distance
```

To register the messages, one needs to use the function *registerMessageCallback*. It takes a messageID, the onDistance function and message type from opendlv_standard_message_set_v0_9_10_pb2 file as input. The messageID was presented in section 5.1 and the opendlv_standard_message_set_v0_9_10_pb2 is OpenDLV standard message set and needs to be imported in the beginning of the program, by using the keyword *import*. The function *connect()* connects the network sessions.

```
messageIDDistanceReading=1039

session.registerMessageCallback(messageIDDistanceReading,onDistance,
opendlv_standard_message_set_v0_9_10_pb2.opendlv_proxy_DistanceReading)

session.connect()
```

A part of an example program could look similar to this example

```
distances = {"front":0.0, "rear":0.0};
```

```
def onDistance(msg, senderStamp, timeStamps):

        if senderStamp == 0:
                distance["front"]= msg.distance

        if senderStamp == 1:
                distance["rear"] = msg.distance



session = OD4Session.OD4Session(cid=112) #112 is cid for kiwi car

messageIDDistanceReading=1039

session.registerMessageCallback(messageIDDistanceReading,onDistance,\
opendlv_standard_message_set_v0_9_10_pb2.opendlv_proxy_DistanceReading)

session.connect()
```

5.3 Send messages

To send data we must first create a message type. This is done by typing opendlv_standard_message_set_v0_9_10_pb2.opendlv_proxy _"*The message type*". For example

```
angleRequest = opendlv_standard_message_set_v0_9_10_pb2.opendlv_proxy_GroundSteeringRequest()
```

We then assign the groundSteering to the correct instance variable as explained in the onDistance function. To send the message we use the function *send* that takes the message ID and the message.*SerializeToString()* as input

```
angleRequest.groundSteering = groundSteering

session.send(1038, angleRequest.SerializeToString())
```

A part of an example program could look similar to the example below

```
angleRequest = opendlv_standard_message_set_v0_9_10_pb2.opendlv_proxy_GroundSteeringRequest()

angleRequest.groundSteering = groundSteering


session.send(1084, angleRequest.SerializeToString())
```

An example of how the skeleton of the code can look like can be seen in this link. Note that the code needs to be further improved and variables changed.

Answer question at related to section 4.1-4.3 https://docs.google.com/forms/d/e/1FAIpQLSd8cyMyqI9yJEA8_N7q5

5.4 Why do we use docker?

In the upcoming section we will introduce docker to build containerized computer environment for our code, that is an isolated virtual computer on your PC for example. The most valuable

advantage of using docker is that we in each container include a small operating system, so if we know that it works in our containerized environment on our computer we also know that it will work on any computer, since the operating system is always included. In this process we have two very important files.

```
Dockerfile
.gitlab-ci.yml
```

The *Dockerfile* is a recipe for how to build out images that we later on want to run on our car. An *image* is a file that execute the code. The recipe contains for example information about that operating system we are using and which programs we must install to be able to run out program. One benefit with dockerfiles is that they can be kept relatively small since we only include the program we need for this image. For more information about dockerfiles, check out this website.

The *.gitlab-ci.yml* file is a description of how the code should be executed and transformed between different platforms. This file is not necessary when we build our containers at our computer but is critical when we later on will push the code to Gitlab. Since the filename begins with a dot, we can't see it in our folder except if we us the flag -a after ls, meaning that we are listing all files in the folder.

In this section we are barely scratch the surface of what one can do with docker. To find out more checkout this website.

5.5 Compiling code with docker

Since we want to run the code on the Kiwi car in a later stage, we want to use docker to build and run our program. If the text is italics it means that it will be replaced by a file, tag or similar.

```
docker build -f Dockerfile -t tag .
docker run --rm -ti - -net=host tag
docker-compose -f file up
```

1. *docker build* builds the image from the Dockerfile. The flag -f specify the name of the Dockerfile, if the flag is not used docker build will search for a file named Dockerfile and run that file. The dot in the end is important and specify that you search for the dockerfile in the same folder that you are currently in.
2. *docker run* is the command to run the image build with docker build command.
3. *docker-compose* replays data into the simulation. -f specifies which file we want to use and *up* starts the replay. How to use this command will be exemplified in section 11.

How to execute the docker build command can be seen in this link. The flag -f specify which file that docker build should run and -t tags the image with the given name.

Answer question related to section 4.4 and 4.5 at `https://docs.google.com/forms/d/e/1FAIpQLSe5tEHRZvnG2-hwY`

5.5.1 Sources of error

I can't build the image using docker build

1. Are you standing in the correct folder, where the Dockerfile exists?
2. Do you have the dot in the end of the command, e.g "*docker build -f Dockerfile* **.**" .

I can't find my images

1. By typing *docker images* one can see the exiting images.

6 First programming task

Use the four sensors (front, rear, left and right) at the car to measure the circumference of a room. You can use the examples above to get inspiration for how to receive and send data, but you will also need to improve the code with cases to deal with incoming data. Use the docker commands to verify that your solution can be compiled without errors. You can hardcode the variables that you need, but for an extra challenge read the necessary input values directly from the terminal (one of them is for example the cid). In the second task you will run the code on the Kiwi car.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSfTfPs--KTEt9lLr18bGqwhS4eELq7ZNQKbVZ9p`

7 Push and compile code on Gitlab

To transfer the code to the Kiwi car we need to push it to a repository. In these tutorials we will use Gitlab.

To push the code we will use four commands

```
git status
git add filename
git commit -m "message"
git push
```

1. *git status* marks all the updated files from last push in red.
2. *git add* is used to specify which files you want to push. If you want to push all files the flag -A can be used instead of the filename, which include all updated files.
3. *git commit -m* creates a new commit with the files added with git add. The flag -m gives you the opportunity to give the commit a message. For example "Added conedetection".
4. *git push* pushes the code to the repository.

All the steps can be seen in this link.

7.1 Working in different branches

In each repository one can use branches to divide the code into different folders. When pushing new code updates, only the current branch will be updated. To create and switch branches we use the following commands

```
git checkout -b branchName
git branch
git checkout branchName
```

1. *checkout -b <BranchName>* creates a new branch with the given name
2. *git branch* shows all the exisiting branches
3. *git checkout <BranchName>* change to the branch called <BranchName>

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSeCbMBpC54KEXXRZ3bTrSmHZD_9kv95IfTFGz7g`

7.2 Continuous integration and deployment

To transfer the code to the Kiwi car we will use Gitlabs CI/CD (*Continuous integration and deployment*). For more information about CI/CD click here. The .gitlab-ci.yml presented in section 5.4 is the recipe for how the code should be executed, in the same way that we used a dockerfile to build our images. The .gitlab-ci.yml gives us the opportunity to build an image for different platforms with the command buildx. To activate the runners needed for CI/CD go to settings → CI/CD → Runners and verify that *Enable shared runners for this project* are accepted. The steps can also be seen here.

7.3 Tag the release

When putting a tag on a push, a new job will be created at the pipeline. Simplified one can say that the pipeline is a workspace where Gitlab can compile the code. To find out more about the pipeline check this website. If the job is successful an image can be downloaded to run the program on the Kiwi car.

```
git tag v1.0 BranchName
git push origin v1.0
```

1. *git tag*, set a tag version on the image and specify the name of your branch.
2. *git push origin*, pushes the image with the tag version given with git tag.

How this is done can be seen in this link.

7.4 Pipeline jobs

Using Gitlab we can see our current and previous jobs. Entering CI/CD in the tab of the left side of the screen and pressing at jobs we can see our previous jobs and their status. One can troubleshoot a job by pressing at the red status button. All the passed steps and at which step the job failed will be shown. This can also be seen in this link.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSdfR3y1y01CizfUSIRbLbIEYKisF0G12PmYaJMW`

7.5 Sources of error

I can't run the git commands

1. Are you standing in the right folder? You can only push code in the cloned folder and the folders underneath it.

The job is pendling

- One reason may be that you have not assign any runners. In the .gitlag-ci.yml you can specify the runners with -tags. Check the names of the existing runners Settings → CI/CD → Available shared runners

8 Downloading the image to the Kiwi car

When downloading the image you need to be connected to the Raspberry Pi as described in section 3.

```
docker login registry.gitlab.com
docker pull registry.gitlab.com/username/repository:tag
docker run --rm -ti --privileged --net=host image:tag
```

1. *docker login* is used to sign in to docker to be able to download the image.
2. *docker pull* downloads the image to the Kiwi car.
3. *docker run*, runs the program at the Kiwi car.

Note that the docker run command can contain different flags dependent on what is required in the image. An example of how it can be done can be seen in this link.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSfn6sDwAAFffOEXkhZK9vjofqBdzQs-xREIhg5f`

8.1 Sources of error

I can't login

- Have you written the correct address? Depending on which registry you are running at there are different addresses. At this link you can read more about container registry.

I can't find the correct image

- You can find all your images by running the command docker images.

When I try to login in I receive the error message" Cannot autolaunch D-Bus wothout X11 $DISPLAY". Video instructions for how to solve the problem can be found link. Note that the video in paused when the installation is running.

- Install gnupg2 and pass at the Raspberry Pi by typing *sudo apt-get install gnupg2 pass* in the terminal.
- Run the command *gpg2 --full-generate-key* to access the necessary keys.
- Choose your options for the keys.

9 Second programming task

Use the information presented about how to push the code, compile it on Gitlab, download the image to run your code from the first task at the Kiwi car.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSejsg4TFd7SjHZiXaQOx3edhC3Bi4izZQZt4gqwm`

10 Image detection

In the second part of this tutorial we are going to turn our attention to image detection using the camera on the Kiwi car. In this section two different ways of working with cone detection will be presented. The first including reading the image directly from the Kiwi car using od4Session. Since the camera updates frequently we will introduce how we work with threads and mutex. This is the preferable way of working with the tutorials and the final task in the section will require this. If you initially want to focus on image detection you can use opencv VideoCapture to replay frame by frame and later on modify your code to read the images form a od4 session. Both these ways of working with image detection will be presented. The material essential for both task will be marked with a green asterisk in the headline while a red asterisk will indicate that the

information is related to working with a od4 session. Copy the same repositories as for the first task and write the code in myApplication.

11 Data replay *

Using data replay we can simulate the camera feed from the Kiwi car and use it to develop our algorithm. Instead of pushing the updates to Gitlab, compile it before pulling the resulting image to the Kiwi car, we can test out algorithm in the simulation to save time. To do so we need to download some recordings. One can find eight recording files here.

1. Create a new folder named *kiwi-recordings* and make sure that the path is $HOME/kiwi-recordings.

If another path is created the last step will not work.

2. Type *wget https://raw.githubusercontent.com/chalmers-revere/opendlv-tutorial-kiwi/master/opendlv-perception-helloworld-python/h264-replay-viewer.yml -O h264-replay-viewer.yml* in the terminal.

The command downloads the file we want to use to replay the data. The file will be located in the folder your are currently standing in.

3. Enter the *h264-replay-viewer.yml* and replace the text *yourFile.rec* with one of the files in the recordings folder.

4. Type *xhost +* in the terminal.

This commands allow us to get access to the running servers and need to be retyped every time Ubuntu is restarted. The folder you stand in when typing the command is irrelevant.

5. Type *docker-compose -f h264-replay-viewer.yml up* to start the container that replays the data.

In the same way docker-compose -f h264-replay-viewer.yml down can be used to stop the container.

All the steps can be seen in this link.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSckRnge4vqeu0E-wYyiDdhLlzmS7h2seQaCo7Gm`

11.1 Input from terminal *

This section is only relevant if you are using od4Session to collect your video data

When running our code on the Kiwi car it is important that we can handle commands from the terminal regarding the cid or height and width of the image for example. Using the package sys we cant easily check the input arguments

```
import sys
print("input arguments: " + str(sys.argv))
```

Running the command python3 myProgram – –height=480 the output will become "input arguments: ['myProgram', '– –height=480']".

11.2 Threads and mutex *

This section is only relevant if you are using od4Session to collect your video data

When reading images from the od4Session we may not be able to process the detection of the cones before a new image is sent. To send the data between the camera and our listener, shared

memory is used. Shared memory handles large volume of data for example video data. To be able to still perform image detection on the data we must lock our shared memory and extract the frames we want to use for image detection. To set up the environment for shared memory we must use the same name as specified in the h264-replay-viewer.yml file or the argument passed to the kiwi by the terminal. You can find the name by searching for the command – – name. First we must obtain the keys to shared memory and semaphores. Semaphores is a data type to control access to multiple processes resource.

```
keySharedMemory = sysv_ipc.ftok(name, 1, True)
keySemMutex = sysv_ipc.ftok(name, 2, True)
keySemCondition = sysv_ipc.ftok(name, 3, True)
```

Next we must initiate the shared memory and semaphore objects

```
shm = sysv_ipc.SharedMemory(keySharedMemory)
mutex = sysv_ipc.Semaphore(keySemCondition)
cond = sysv_ipc.Semaphore(keySemCondition)
```

When extracting data from the shared memory we have five commands that is essential.

```
mutex.acquire()
shm.attach()
buf = shm.read()
shm.detach()
mutex.release()
```

1. *mutex.acquire()* locks the access to shared memory
2. *shm.attach()* attach the shared memory
3. *shmbuf = shm.read()* read shared memory into our own buffer
4. *shm.detach()* detach to shared memory
5. *mutex.release()* unlock access to shared memory

When using shm.read() in the third step the data is not in the correct format. To transform the data, so it can be used with opencv, we must use the following command.

```
img = numpy.frombuffer(shmbuf, numpy.uint8).reshape(720, 1280, 4)
```

The data is collected from the buffer and transformed to an 8 bit integer with size 720x1280 and with the four colors (RGBA) red green blue and alpha. More information and other methods that can be used for shared memory and semaphores can be found in this link.

Answer question at https://docs.google.com/forms/d/e/1FAIpQLSdhYOdBYDuA76UjJxngCt7K9qDnlWCtUi0xX18s

12 Cone detection *

An example of how your detection can look like after this step can be seen in this link. In the following steps we will use the variable name *frame* as the image we will work with and mainly using the package cv2 (opencv). First we see if we have any video capture or any od4Session (preferable) that is running. Depending on if you are running a od4Session or a video capture the syntax will look a little different.

```
# Video capture

cap = cv2.VideoCapture('Your_file')
```

```
while(cap.isOpened()):

    ret, frame = cap.read()

# od4Session, where your od4Session is defined as od4
while(od4.isRunning()):
```

Secondly we want to transform out color space from blue green red (BGR) to hue saturation and value (HSV) since this will simply the detection. This is done with the method cvtColor in cv2.

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

To select which colors we want to detect we must specify a range. For the yellow cones in the video the following lower and upper limit was chosen. Note that mixing with the intervals may result in an better detection in some environment but worse in different depending on external sources such as light.

```
low_yellow = np.array([18, 94, 140])
upper_yellow = np.array([48, 255, 255])
```

To apply the limits we use the method inRange where we specify the frame we are using and the lower and upper limit of the colors we want to detect.

```
mask = cv2.inRange(hsv, lower_blue, upper_blue))
```

To improve our detection we will use dilation and erosion. Dilation is used to "grow" the image to clog a miss detected pixel in the image. While erosion is doing the opposite and "shrinking" the image. By first applying dilation we can expand out image and fill the gaps of miss detected pixels within the cone for example. Dilation will also expand the outer boundary of the cone. By using erosion we can reverse the expansion by shrinking the outer boundary of the cone. If you would like to read more about dilation and erosion and the mathematical operation behind the methods in opencv, a good starting point is to check out this website. There is more methods in opencv that can be used to further improve the image detection. We wont explain these in this tutorials but for the curious one check out this link.

When working with dilation and erosion we must define a kernel deciding how many pixels that are affected during the operations. What the kernel does can be seen in figure 4. Starting with our modified frame to the left, where 1 is indicating a detection and 0 no detection, the kernel is defined with its origin in the lower left corner and ones at position (1,1), (2,1) and (1,2). The kernel sweeps through the picture starting in the upper right corner. If a one in the kernel is matching with a one in the frame the resulting picture after the dilation will have a one in the same position in the frame where the origin of the kernel is currently positioned. When the kernel is moving its way though the frame more ones will most likely be added and the picture will grow. The kernel will have the opposite effect if it is applied in erosion. If not all the ones in the kernel matches with ones in the frame, the pixel where the origin of the kernel is currently in will be set to zero.

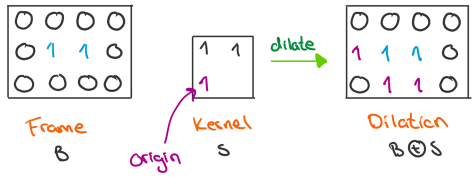


Figure 4: Explanation of dilation using a kernel

As a starting point I would recommend a 3x3 kernel with ones.

```
kernel = np.ones((3,3), np.uint8)
```

When using dilation and erosion in opencv it takes three arguments, a frame, the kernel and the number of times the operations should be applied.

```
dilation = cv2.dilate(mask,kernel, iterations = 1)
erosion = cv2.erode(dilation,kernel, iterations = 1)
```

After showing the images we must have the following lines. The first statement in the if statement is a method that is waiting for 40ms before breaking the current frame, so we can show a new one. The second argument compare helps us terminate the replay. Pressing q the input will be compared with the ascii value for q and the program till break. Outside the if statement we release the video capture and then close all windows.

```
    if cv2.waitKey(40) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

An example program can be found in this link. Make sure that you have the recording files in the same folder.

Answer question at https://docs.google.com/forms/d/e/1FAIpQLScGZyTkB6R7ESKnNeadqGClRE6lQSro_I0BhSZOI

13 Find position of the cones *

In the next step we want to find the position of the detected cones and use it to estimate the roadside. In figure 5 an illustration of how estimated roadside can be seen.

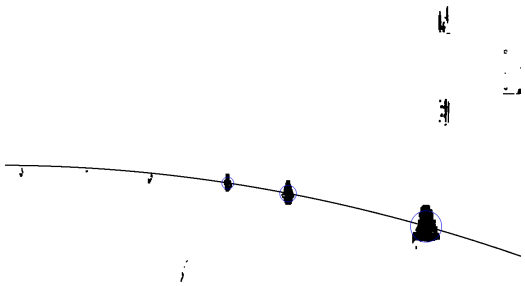


Figure 5: Illustration of how to detect cones and how to draw a curve representing the roadside.

To detect the cones one can use the *SimpleBlobDetector* available in the cv2. In section 5 the detected cones were white and the background black. Inverting the colors will help us later on when we want to plot objects in the picture.

```
imagedata_original = 255-imagedata_originalRevers
```

since the RGB for white is (255,255,255) while for black (0,0,0). Next we must create our SimpleBlobDetector object.

```
params = cv2.SimpleBlobDetector_Params()
```

By setting the parameters we can adjust what we will detect as cones. We may for example only be interested in the nearest three cones when we are calculating how the car should steer and therefore setting a lower limit on the detected blobs area. Here you can find information about the SimpleBlobDetector class and the parameters. To create the SimpleBlobDetector with our parameters one types

```
detector= cv2.SimpleBlobDetector_create(params)
```

and to get our detected keypoints one runs

```
keypoints = detector.detect(imagedata_original)
```

, where imagedata_original is our image. Using the method .pt on the object one can extract the coordinates of the keypoints. These coordinates will be used to estimate the roadside.

```
from numpy.polynomial import polynomial as P
c, stats = P.polyfit(keypointsX,keypointsY,2,full=True)
```

By importing polynomial we can fit a n-degree polynomial function to our data points. In the example above the coefficients of a second order degree polynomial function is calculated from our keypoints. In order to plot the keypoints and the polynomial function in the same frame as the original image we can use opencv's existing methods. Three methods that can be used are

polylines
circle
drawKeypoints

In this link you can find out more about Opencv drawing functions and in this link how to draw functions of keypoints.

An example code for can be found in this link. Either use your own images our you can download these image, image1. The second image is more challenging to detect cones in.

Answer question at https://docs.google.com/forms/d/e/1FAIpQLSewtBBi2TtMOZvDnTTSQPziC80_JPrVcv3ddB7t

14 Task 3 *

Detect the cones and draw a line similar to the one in figure 5 representing the roadside. Update the detected cones and the roadside as often as your program can. If you want to challenge yourself, use data replay presented in section 11 otherwise replay the data with VideoCapture.

14.1 Hints and problems to consider

1. Depending on the background and color span one may get false detection. Consider to only detect objects in the area of the cones.
2. If no cones are detected how should we avoid that the program crashes?
3. How can the parameters on the SimpleBlobDetector be optimized to work in general situations?

Answer question at https://docs.google.com/forms/d/e/1FAIpQLSdAOFiwQQleOOdnuQBg5CPtWxPSbWa4UKJGjjed

15 Task 4 *

Extend your code from task 3 so that the Kiwi car can follow the lane without hitting any cones.

15.1 Hints and problems to consider

1. what will happened if the code running at the Kiwi car contains prints or plots?
2. How can the steering algorithm be optimize to run both at a straight distance when one can detect both yellow and blue cones and in a curve when we may only be able to detect one of them?
3. How will the car react of no cones are detected?

4. Which inputs argument are necessary? For example it may be good to know the height of the image illustrated in section 11.1.

Answer question at `https://docs.google.com/forms/d/e/1FAIpQLSc0xmRWRTAHszG4ymkGyVM4KQgPzDAXV0fW4AcgP`

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY