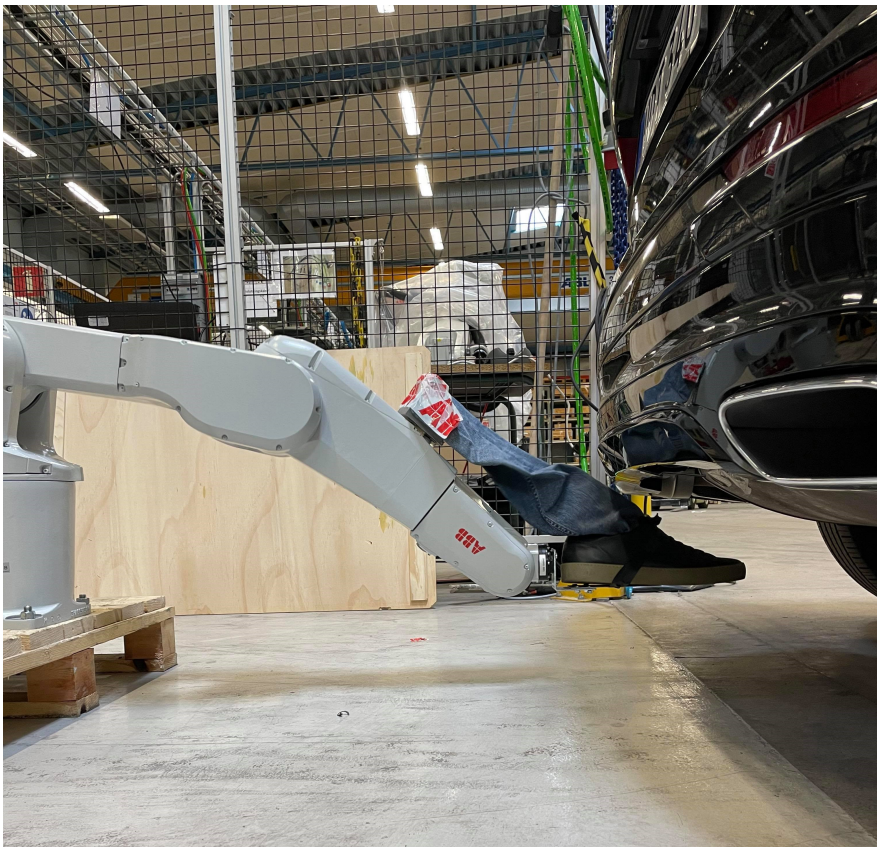
CHALMERS
UNIVERSITY OF TECHNOLOGY



Human body movements performed by machines

Master's thesis in Product Development

FREDRIK KUYLENSTIERNA

DEPARTMENT OF INDUSTRIAL AND MATERIALS SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

A master's thesis that aims to record and mechanically reproduce human body movement

FREDRIK KUYLENSTIERNA

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Materials Science
Chalmers University of Technology
Gothenburg, Sweden 2021

A master's thesis that aims to record and mechanically reproduce human body movement
FREDRIK KUYLENSTIERNA

Supervisor: Hanna Börjesson, Volvo Cars
Examiner: Asst. Prof. Karinne Ramirez-Amaro, Department of Electrical Engineering

Master's Thesis 2021
Department of Industrial and Materials Science
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The developed final prototype of the project performing a kick under the bumper of a Volvo XC60.

Typeset in LaTeX
Gothenburg, Sweden 2021

A master's thesis that aims to record and mechanically reproduce human body movement

Fredrik Kuylenstierna
Department of Industrial and Materials Science
Chalmers University of Technology

Abstract

Recent customer feedback received by Volvo Cars shows that the function to open the tailgate without using any hands is among the top functions on the vehicle subject to user complaints. The current test and calibration methods of the function involves repetitive kicking by a human to open and close the tailgate. The aim of this project was to analyze a human kicking motion and recreate the same motion with a robot, which in turn should activate the opening and closing of the tailgate. By replacing the human kicker with a robot, unwanted variation of the kicking motions in the calibration and test processes is eliminated.

A method to record human kicks with an optical motion sensor tracking technology has been developed. The motion tracking uses a VR environment in the Unity software as a platform to generate Cartesian data from the movement. The generated motion data is imported to the software RobotStudio to enable offline robot programming. A trajectory is created in RobotStudio, based on the input file of the recorded human motion data. The trajectory is then performed by a real robot of the model IRB1200 from the manufacturer ABB Robotics. The robot was outfitted to resemble a real leg by attaching a shoe and jeans material.

The functionality of the solution was verified with a Volvo XC60 through quantitative testing. The testing confirms that the robot can successfully repeat identical kicks to open and close the tailgate. Future work for Volvo Cars includes developing new test programs that utilize the solution.

Keywords: Human motion tracking, Robot movement, Virtual reality applications, Robot trajectory planning, Human leg replication

Acknowledgements

First and foremost, I would like to express my gratitude for my project partner Joël Brou Boni. We had a great collaboration throughout this project and performed all practical development activities as joint efforts.

I would like to thank the department of *Door Control, Locking & Alarm* at Volvo Cars for the opportunity to write this master's thesis. In this department, I would like to express a special thanks to my supervisor Hanna Börjesson for your supervision, support and help throughout the project. I would also like to thank the department *Open Innovation Arena* at Volvo Cars for the support during the development of the motion tracking solution. A special thanks to Timotei Florin Ghiurau, Aljoscha Ledwa and Romit Godi working in this department. I would also like to thank the persons supporting the project from ABB Robotics. This includes Göran Manske, Patrik Nilsson, Viktoria Lundstöm, Anders Spaak and Göran Bergström who provided the project with a robot and supported the creation of the final prototype. Finally I would like to thank my examiner Karinne Ramirez-Amaro for the support during the development process and for helping me succeed with the project.

Fredrik Kuylenstierna, Gothenburg, June 2021

Table of Contents

1
Introduction

The modern automotive industry is transitioning into a new era. The car ownership in 2021 is no longer just about transportation. The focus has shifted towards communication, connectivity and convenience and the user is at the center of it. Car manufacturers all over the world spend a lot of time, money and effort on coming up with new concepts and functions with the intention of elevating the user experience. A function that achieves this is one where the user can perform touchless opening and closing of the tailgate on the vehicle. This allows for access to the trunk without using any hands. One situations where this is useful is when returning from the grocery store with one bag in each hand. The touchless opening allows the user to load the bags into the car without ever placing the groceries on the ground. A second scenario when the function is helpful is when the tailgate is dirty and the user can perform the opening without having to worry for becoming dirty. One common way to activate the function is by performing a kicking motion underneath the rear bumper. This feature is used by many car manufacturers for models identified in the premium segment. Volvo Cars is a company acting in the premium segment and includes the touchless opening as an option for most vehicle models [1]. However, the feature is known for receiving user complaints from a range of different car manufacturers. This master's thesis was a proposal from Volvo Cars to generate a tool that will enable better testing methods of the unit performing the opening and closing of the tailgate and thus support its future development.

1.1 Background

Recent customer feedback received by Volvo Cars shows that the function to open the tailgate without using any hands is in need of modifications and upgrades. It is among the top functions subject to customer complaints [2]. There are two prominent complaints about the function. The first one being that there is no clear indication on the car where to perform the kick to open the tailgate. As there is no indication, many users perform the kick in the center of the car. The sensor that triggers the opening mechanism is positioned more to the left when facing the rear of the car. This results in that the user is kicking further away from the sensor that what sensor is able to register as a kick. Since no kick is registered by the sensor, the opening mechanism does not activate on the car. The second uncertainty received from the customer feedback is which type of kick that should be performed to activate the opening mechanism. The sensor setup used on the car allows for a range of different types of kicks to trigger the sensor. One requirement from the

sensor to register the approaching motion of a foot as a kick is that there needs to be a distinct forward motion reducing the distance to the sensor, followed by a distinct backward motion increasing the distance. One common mistake by unaware users is trying to activate the opening mechanism by performing repeated kicks in close proximity to the bumper or sweep the foot along the bumper of the car. A motion like this is registered as a correct approach to the sensor, but due to the lack of distinct backward motion of increasing the distance to the sensor, it is not registered as a kick.

Today at Volvo Cars, the function's sensor setup is calibrated manually by engineers in-house. The calibration involves repetitive kicking to open and close the tailgate. An evaluation of the calibration values for the sensor is made based on scenarios where the sensor is activated, or not activated, by a kick. As the calibration process involves human interaction, there is a risk for variance between different units. In addition to this, the rear bumper design differs between different models in Volvo Cars' vehicle line-up. To enable objective calibration of the function, this master's thesis will investigate how to mechanically reproduce a human kicking motion and thus eliminate the need for any human factors in the calibration process. Doing so will also allow for future development of the function and benchmarking of the same feature on competitors' vehicle models.

1.2 Related work

The project consists of two main areas; motion tracking and robot movement. In order for the developed solution to be a credible replacement of a human during the testing, the robot needs to perform equivalent kicking motions when trying to open the tailgate. The approach to achieve this was by finding a method to perform motion tracking of a human kick, followed by generating robot movement based on the recorded motion data.

1.2.1 Motion tracking

One study dealing with human motion capture is made by Nakamura, Lee and Ott [3]. The authors propose a Cartesian control approach for the motion capture. In the study, a set of markers were placed on the human's upper body and the positions were being monitored. These markers were connected to a corresponding point of the humanoid robot, by using virtual springs. The forces acting on the virtual springs will then generate information about the desired motion of the robot. The positions of the lower body in the model was generated by a balancing control algorithm, taking the constrains of the full body pose into consideration when calculating the joint angles. A whole body motion was generated with this method.

Jung, Kim and Lyou [4] propose a hybrid motion caption system, using both optical and sensor type motion capture devices. A multiple camera setup is calibrated to obtain positions of markers placed on a human body. Then, joint rotation information was obtained by attaching multiple sensor modules to to the human body, legs

and arms. This generated data for human skeleton reproduction. Advantages to this approach is that multiple persons may take part in the motion capture, as the two systems act complementary.

1.2.2 Robot movement

An area of interest for the project was offline programming of robots. Offline programming refers to the method of generating robot programs on an external PC, outside of the physical production process where the robot is normally placed. Studies have been made of how to increase the speed and efficiency of how a desired production process can generated. One study by Holubek et al. [5] explores how imported computer aided design (CAD) models can be used as a tool in the offline robot programming. The study uses a virtual ABB robot, of model IRB140, along with ABB's own software RobotStudio where the virtual environment is created. It is described how an imported CAD model can be utilized for creating targets by using the surface of the geometry. Based on the targets, paths can subsequently be created by commanding the robot to pass through the targets.

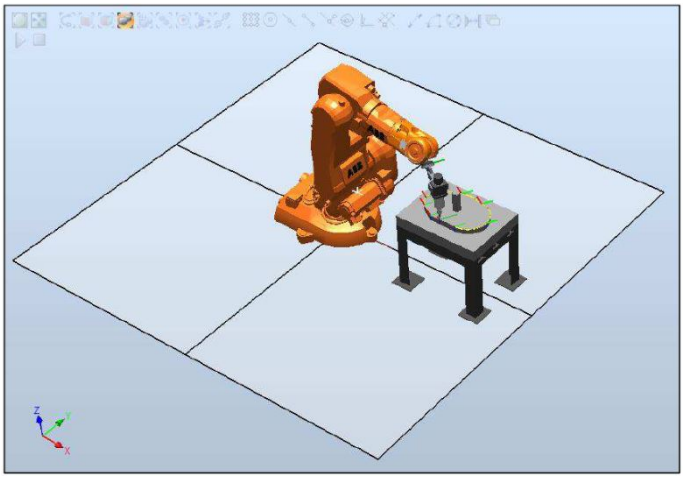


Figure 1.1: Work station in RobotStudio with a generated trajectory from an imported CAD model [5].

Emphasis is placed on the quality of the CAD model being imported. The authors of the study use CATIA V5 for modeling of the geometry. The process of creating robot programs in RobotStudio is described as a complex series of related sequential steps, where the best approach is said to be creating targets and paths.

1.3 Aim

The aim of the master's thesis was to analyze a human kicking motion and mechanically recreate the motion with the intention to activate the automated function of opening the tailgate on a vehicle from Volvo Cars. In order to achieve this, the project had the following objectives:

- Find and evaluate a suitable method for collecting data on body movements.
- Find and evaluate a suitable method for mechanically reproducing human body movements.
- Record human body movements with a suitable motion capture sensor technology.
- Reproduce the recorded body movement using a suitable robot.
- Verify the data collection and the corresponding robot movement in relation to a real vehicle.

1.4 Problem definition

In order to mechanically recreate a human kick, there were a number of problems that needed to be solved. The initially known issues that the project dealt with are presented below.

- Which motion tracking method is the most suitable to accurately record human motion in three dimensions?
- Which robotics solution is able to best recreate the motion of a human kick?
- How should the data from the recorded human motion be stored to allow the captured movement to be imported as a trajectory the chosen robot?
- How should the robot be adapted to constitute a satisfactory representation of a human leg in terms of material and geometry?

1.5 Research approach

The project adopted a structured approach to find solutions to the named objectives in Section 1.3. Table 1.1 below outlines how each objective was managed.

Objectives	Research approach
Find and evaluate a suitable method for collecting data on body movements.	By a literature search find and present existing human body movement tracking techniques. The search covered applications in different industries and with varying use cases.
Find and evaluate a suitable method for mechanically reproducing human body movements.	A literature search was conducted to explore available methods of how to reproduce movements similar to those of a human. The search included an investigation of how to map recorded data of human movement with the desired movement of a robot.
Record human body movements with a suitable camera or motion capture sensor technology.	Based on the literature search, human body movements was recorded with the method found to be most suitable for the project. Which use cases to record was decided in agreement with Volvo Cars.
Reproduce the recorded body movement using a suitable robot.	The investigation of how to map recorded human motion to mechanical movement resulted in a procedure of how to reproduce the movement. The usability of the chosen robot was verified.
Verify the data collection and the corresponding robot movement in relation to a real vehicle.	Testing in its intended use application was required to verify the functionality of the final prototype. The testing included adaptation of the prototype to achieve suitable sensor activation on the vehicle.

Table 1.1: Research approach

1.6 Foot Movement Detection Module

The unit responsible for the automated opening and closing of the tailgate is called the Foot Movement Detection Module. Internally at Volvo Cars, it is known as the FMDM. This project was proposed from Volvo Cars in order to support their development of the unit. The final prototype of the project is intended to be used during calibration and testing activities by engineers working with the function. The prototype will be used as a tool to activate the sensors of the module. The project outcome will thus indirect promote the future development of the function by enabling new test procedures of objectively repetitive kicking motions performed by a robot.

In order to detect and execute on triggering motions by an approaching object, the FMDM operates by a combination of motors and sensors. Due to a non-disclosure agreement with Volvo Cars, no further information of the functionality or specification of the module will be included. However, essential information of the triggering sequence is that the sensors will react to the following parameters of an approaching object:

- Speed
- Material
- Geometry

These parameters constituted the basis for the product development processes of achieving correct sensor activation with the prototype. Out of these parameters, the speed was the main focus. The main reason for this being that the functionality of having a sufficient speed for sensor activation was a feature that would be

problematic or not even possible to add to the final prototype in the later stages of the project. The process of adapting an appropriate geometry and used material for the final prototype is more flexible and the parameters are thus seen as secondary to the speed.

1.7 Demarcations

The main objective of the project was to develop a prototype that can demonstrate the function that was asked for by Volvo Cars in the project proposal. Thus, there was no intention to developed a product ready for the outer market. No potential external customers were be taken into account when developing the prototype. The requested function was one where a robot is able to receptively and with high accuracy reproduce identical kicks to those previously performed by a human. This entails that the project covers two main areas, motion tracking and robot movement. Regarding the motion tracking, there should not be any restrictions of which type of kick is being recorded by the operator. This means that when the operator is placed behind a vehicle and performs a kick, the full trajectory of the motion should be captured. However, practical factors have to be considered in order to ensure satisfactory recreation of the kicks. If the robot that will recreate the kicks is stationary placed behind the car, the reach of the robot, i.e. the total length of the robot when fully extended, will be a limiting factor. Some consideration of the length of the recorded trajectories will thus have to be made. The second area is the robot movement. The function asked for by Volvo Cars is a robot that can produce repeated kicks, identical to each other when objectively compared. This will be used as a tool internally to develop the FMDM function through different test activities. The scope of the project includes to create the tool. Thus, the project does not cover the process of developing the FMDM function itself by creating and performing the activities where the tool is used. Also, the project will not make any efforts to communicate or read the sensor output of the FMDM sensor. The only interaction with the car and the FMDM sensor will be performing kicks underneath the bumper with a robot, where the outcome can be either a successful or unsuccessful openings of the tailgate.

As stated in the scope for the master thesis project, the intended time period is one semester, representing 30 credits and approximately six months [6]. There was no available pre-set budget for the project from either the university or the company. This resulted in that some consideration were made to cost and time during the project to ensure the expected completion. One decision where this became relevant was when procuring equipment for the motion tracking techniques. The project was able to procure the desired equipment from within the Volvo Cars organization. This saved both cost and time compared to procuring similar equipment from an external source.

1.8 Additional aspects of the project

In order to ensure that the project work and all its outcomes maintained the highest professional and ethical standard, the IEEE Code of Ethics was considered throughout the project [7]. The list of ten guidelines represents a framework of how to act as a professional. One prominent item on the list for this project was the following:

"3. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;" [7].

The project work was conducted at Volvo Cars. This means that the author had a responsibility both towards the company and Chalmers University of Technology. One issue that made this aspect relevant for the project work relates to the non-disclosure agreement that was signed with Volvo Cars. The agreement prevents an in depth presentation of the FMDM unit. As this was the main component of the car that the project work focused on, a compromise had to be done to include sufficient information about the FMDM unit to understand the background and the implementation of the project without exceeding the boundaries of the non-disclosure agreement.

1.8.1 Social aspects

As stated in Section 1.3, the aim of the project was to mechanically recreate a kick performed by a human. This in turn will result in an operation where a task previously performed by a human is instead executed by a robot and thus the operator gets replaced. One could argue for that performing a large amount of repetitive kicks is not a productive engineering activity. Replacing the human kicker with a robot will release the engineer that is currently performing this task to work with other activities. This brings the potential to promote the development of the FMDM function in better ways that what is currently being performed. The introduction of a kicking robot will also generate a statement of investment towards the development of the FMDM function. It will enable new testing activities for the engineers working with the function, that are currently not possible to be performed. In order to work with the full process of recording human movement and recreating the trajectory with a robot, knowledge about how to use the hardware and software is necessary. Introducing the robot as a tool to the group working with the FMDM sensor will thus generate one or multiple new roles. Having knowledge about the motion recording and robot programming will be added to the list of responsibilities that are distributed within the team.

1.8.2 Ecological aspects

In the short term, introducing a robot to be used in the testing and calibration of the FMDM sensor will have a negative impact on the carbon footprint of Volvo Cars. The resources to manufacture the robot, the energy consumption while using it and the disposal of the robot all need to be taken into consideration during the

assessment. The development work of testing and calibrating the FMDM sensor is usually performed on a single vehicle from each car model, including the different bumper designs for different trims of the same model. One example is the difference in bumper design for a V90 Inscription and a V90 Cross Country. This adds to a few dozen different bumper designs in total where the robot would be used. If the robot is introduced as a tool to calibrate the FMDM sensor, separate calibration would be performed for all models. The values and settings from the calibration are then downloaded to all newly manufactured cars leaving the factories. The hardware inside the FMDM unit also undergoes testing at Volvo Cars, even though some of the lifetime and reliability tests are performed by the supplier providing the module. In 2020, Volvo Cars sold 661 713 cars globally [8]. Not all sold cars include the automatic tailgate opening, but any potential issues related the hardware or software of the FMDM unit could bring a huge carbon footprint impact due to the large number of cars it would affect. If a car is required to visit a dealer to change the FMDM module due to hardware failure or to update the software because of inadequate sensor calibration, the carbon footprint of the robot, that brings the potential to avoid the issues due to implementing better testing methods, will be neglectable.

2
Technical Background

This chapter presents more detailed information about the areas the project will deal with in the later chapters of the report.

2.1 Fundamentals of imitation

As described in 1.3, the motion of a human will be the basis for the succeeding mechanical movement. A description of the event that occurs may thus be that a robot imitates a human. A detailed definition was introduced by Mitchell, where the following requirements needs to be satisfied for imitation to be recognized [9]:

1. Something C is produced by an organism and/or machine, where
2. C is similar to something else M,
3. Registration of M is necessary for the production of C, and
4. C is designed to be similar to M.

By this definition, the human motion represents the model M and the robot movement the copy C.

2.2 From motion recognition to reproduction

An approach to human motion imitation is presented by Kulić [10]. Her work covers a presentation of the key methods for first observing and later modeling of human movement, as well as the reproduction of the movement by humanoid robots. She describes that the processing pipeline of human motion imitation can be divided into four stages.

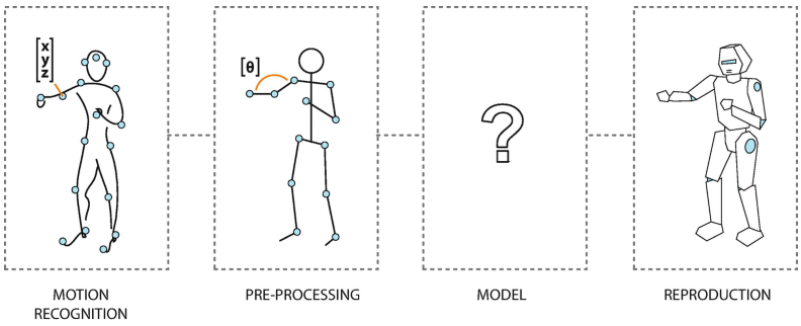


Figure 2.1: The processing pipeline for human motion imitation [10].

The first stage of *Motion recognition* involves observing and recording motion. To achieve the highest accuracy, this is typically performed with motion capture, meaning small markers are placed on a human body. The 3D position of the markers are captured by a set of cameras, resulting in a data set of the Cartesian trajectory (x,y,z positions) for each marker over the duration of the motion. The second stage of *Pre-Processing* involves removing noise, interpolating missing marker data and extracting relevant segments of the motion. In order to convert the Cartesian data set into a kinematic model used by a robot or a human demonstrator, inverse kinematics are typically used. To put this into context, the study of motion where the cause of the motion is not included, such as torques and forces, is known as kinematics [11]. Using the kinematic equations to determine the required motion of a robot to reach a desired position, is known as inverse kinematics. When applying inverse kinematics to the Cartesian data set, a joint configuration with calculated joint angels is generated based on desired end position of the robot or human demonstrator. A *Model* of the movement is created in the third stage. Kulić describes three classes of approaches for generating models:

1. Direct human trajectory reproduction with separate postural control.
    - This approach uses a reference trajectory, containing joint angle time series that are obtained from the motion capture of a human movement. A control strategy is formulated to follow the reference trajectory as closely as possible.
2. Motion primitive-based approach.
    - This approach is based on the notion that human movement can be decomposed to a set of primitive actions, known as motion primitives. The motion primitives are pre-computed motions that the robot or human demonstrator can execute. The approach is then to learn the models to sequentially transition between valid motion primitives from the given state to reproduce humanlike movement.
3. Control policy approach.
    - The aim of this approach is to generate a learning system, whose goal is trying to imitate the control system used to generate human movement. Once a control system, or policy, is known, it is applied to the robot or human demonstrator to generate humanlike movement. One method of generating a control policy is by using reinforced leaning. The reinforced learning framework includes iterative testing with a robot, where successfully executed actions are rewarded.

The forth stage refers to the *Reproduction* of the movement with a robot or a humanoid. This is performed by using the developed model together with additional control elements.

2.3 Computer aided design

CATIA V5 is the preferred computer aided design (CAD) software at both Volvo Cars and Chalmers University of Technology. It is developed by Dassault Systems. The software offers a variety of built-in functionality, useful for a range of design tasks. A function relevant for the project is one where coordinates may be imported to the software and automatically generate geometries based on the input. The function utilizes a pre-configured macro in a Microsoft Excel sheet and is a part of the installation directory of CATIA V5. The coordinates entered in the sheet need to have a positive value in order for CATIA V5 to accept them. The name of the Excel sheet is *GSD_PointSplineLoftFromExcel*. The macro provides three options along with the coordinate import:

1. CreationPoint
    - The coordinates in the Excel sheet are imported as a point cloud. There are no interconnections between the points in the cloud.
2. CreationSpline
    - The coordinates in the Excel sheet are imported as a point cloud. In addition to this, a spline is created between the start and end point in the list of coordinates. A curve fit is used to connect all data points in the spline. Multiple lists can be imported simultaneously and generate individual splines.
3. CreationLoft
    - The coordinates in the Excel sheet are imported as a point cloud, with splines connecting the points. Multiple splines generate a multi-section surface (loft). The surface connects the start and end spline in the list of coordinates. A curve fit is used to include all splines in the loft. Multiple lists can be imported simultaneously but only one loft is generated.

3
Method

This chapter presents the methodology of the project. It was applied in such a way that the objectives of the project and their potential solutions were met in an effective way, in order to reach an appropriate final result. The framework of the methodology consists of a general solution oriented product development. The methodology of the project is presented by the flow chart in Figure 3.1 below. A more detailed description of the tasks are presented in the succeeding sections.

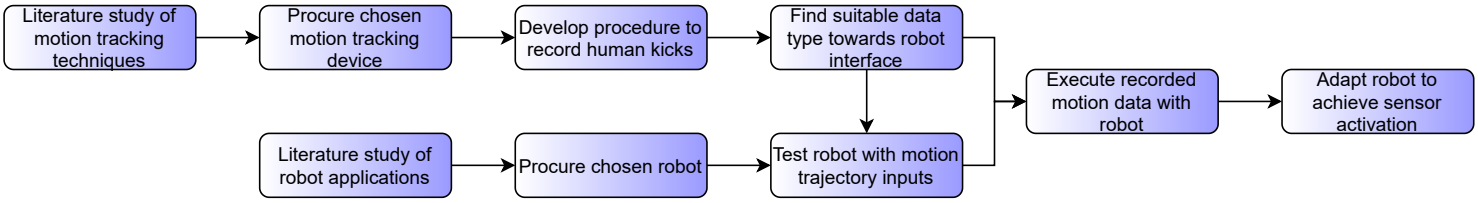


Figure 3.1: Flow chart of the project work

The related work was an inspiration for the development process in the project. By using a Cartesian approach for the motion capture, as suggested by Nakamura, Lee and Ott [3], the output data would be in a usable format to be processed in a CAD software. As demonstrated by Holubek et al. [5], geometries can be used for generating trajectories when imported to an offline robot programming software. When applying the theory to this project, a CAD software could be used as an intermediate step between the motion capture and the robot movement. In Section 3.6.2.1, the method is implemented by using the CAD software CATIA V5. In order to perform accurate motion tracking, Jung, Kim and Lyou [4] proposes to use a hybrid system consisting of two motion tracking technologies. This approach was implemented in the development process of this project as two different technologies were used during the initial testing, described in Section 3.4.1.

3.1 Current solutions

A literature study of the current solutions for motion tracking and robot movement were performed. The study included research projects and existing products on the market.

3.1.1 Motion tracking

Motion tracking can be performed in various ways. What is a shared characteristic between different techniques is the ability to express the location of a point in space at a specific time. This point in space is specified by three Cartesian coordinates, namely *x(t)*, *y(t)* and *z(t)*. If only the position of the point is requested, it is called a 3 Degrees of Freedom (DoF) tracking problem [12]. If the orientation of the point is also considered, three degrees of freedoms are added, resulting in a 6 DoF problem. In order to fully specify the position of a rigid body, both position and orientation is required, which corresponds to 6 DoF [12].

3.1.1.1 Tracking techniques

When describing available systems for motion tracking, they can in a simplified way be categorized into two main areas: camera based and sensor based systems. Among these systems, there are three predominant techniques to capture motion [13]:

1. Video recordings
    - This solution uses a single or multiple cameras to record the motion. With the help of computer vision applications, relevant features are extracted from the video.
2. Optical, infrared motion capture
    - Reflective markers are placed on the moving object. By using multiple infrared cameras, the position of the reflective markers can be determined.
3. Inertial measurement units (IMUs)
    - The units consists of gyroscopes, magnetometers and accelerometers. Together, they can be used to determine the position and orientation of the unit.

Performing motion capture with a single camera primarily limits the solution to two dimensions [13]. A certain camera position may enable horizontal capture of a moving object but it will have limited capability to report the vertical movement from the camera. This limitation can be overcomed by adding multiple cameras. A 3D representation of a motion, recorded by multiple cameras, can be created with data fusion software. Figure 3.2 shows a generated 3D model from two moving bodies recorded with a single RGB camera.

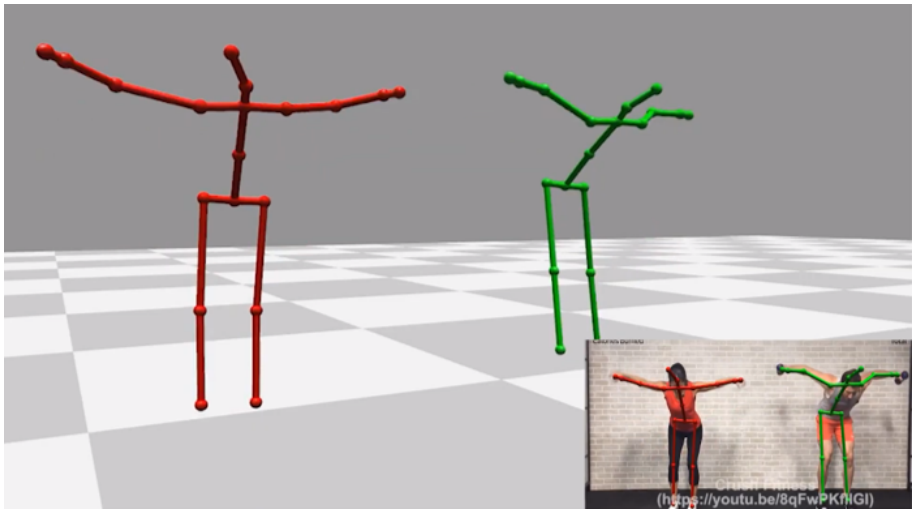


Figure 3.2: Multi-person 3D motion capture with a single RGB camera [14].

Optical motion capture requires multiple cameras and a calibration process of the system before recordings are possible. When used outside any dedicated test environment, this brings a number of problems. It does not allow for instantaneous motion capture as there needs to be an installation process of the system. After the installation, any changes to the camera placement may cause inaccuracies to the motion capture. The area of which objects can be traced is also limited to the number of cameras used and their placement in the test area. The reflective sensors on the moving object needs to be visible for the cameras at all times for complete motion capture. However, with an appropriate test setup, the optical system offers tracking of high accuracy and precision [13]. Figure 3.3 shows a simulated setup of a body with reflective markers standing in the center of a test area, surrounded by infrared cameras.

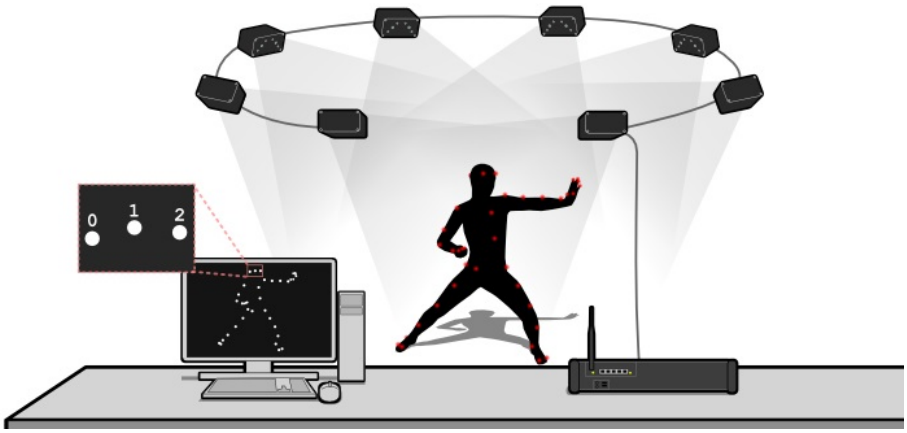


Figure 3.3: Setup for Impulse X2 motion capture system [15].

Inertial sensors overcome many of the challenges posed to the camera based systems. As the sensors are an integral part of the unit itself, it is not limited to a test area surrounded by cameras. The method is therefore beneficial when recording

movements in a real world application. IMUs were included in the test setup for a biomechanical analyzis of a skier through a 10-gate slalom course [16]. Inertial sensors inherit challenges with accurate positioning over longer recordings. The sensors does not measure the position of the unit itself, but the rate of change it is exposed to [13]. A satisfactory estimation of the units position can be made through integration, but over time the position will suffer from drift. The drift in position can be eliminated by adding supportive systems to the unit's positioning, such as a global positioning system (GPS) or an acoustic positioning system [17]. Figure 3.4 shows an avatar with multiple IMUs attached to it.

Figure 3.4: Multiple IMUs attached to an avatar [18].

In this project, only motion tracking through methods using optical infrared motion capture and IMUs were considered. The reason for video recordings being excluded is their inherent difficulty to report accurate positioning in three dimensions. As the motion to be recorded was a kick next to and underneath the bumper of a car, additional visual obstacles would be introduced for the video camera system. In order to accurately reproduce the kick, all three directions of motion will be equally important. The optical infrared motion capture can be adapted so the infrared cameras have full visuals of the human leg even under the bumper. When using IMUs, no external tracking equipment is needed and no limitations are put on the visibility of the leg.

3.1.1.2 Motion tracking in virtual reality

A key component of a virtual reality (VR) system is digitizing human movement and make it usable in a computer software. Proper motion tracking of a human is elementary for a successful system. VR applications utilizing motion tracking are being used in a number of different areas such as gaming, virtual prototyping and training for medical and military scenarios. It is the same techniques as described in the previous Section 3.1.1.1 that enable motion tracking in VR [19]. Optical tracking is performed by using infrared cameras, also known as base stations when used in a VR application, to determine the position of the connected equipment to the VR system. Examples of equipment that can be tracked by the base stations is a VR headset, enabling the user to visually experience the VR environment, or a controller that allows the user to interact with environment. Non-optical tracking techniques attach IMUs to the body being tracked, either as individual units or incorporated in clothing. When incorporated in a full body suit, motion capture for the entire body is generated.

3.1.1.3 Available products for motion tracking in virtual reality

In the current market, there are a number of companies offering motion tracking in VR by different tracking techniques. Products using IMUs include full body suits such as the ones from Xsens [20] and Teslasuit [21]. Included in the purchase of each suit, is access to specialized software developed by the companies. The software enables easy calibration, playback and reprocessing of the motion capture as well as real-time visualization of the movement [20]. The cost of the mentioned full body suits using IMUs for motion tracking is in the region of 100,000 SEK.

One popular system using optical, infrared motion capture to enable tracking in VR is from the company VIVE. The set includes a headset, two base stations and two controllers which allows visualization and interaction with the VR environment. The general recommended computer specifications for running the system on a PC is according to VIVE [22]:

Graphics: NVIDIA® GeForce® GTX 1060 or AMD Radeon™ RX 480, equivalent or better.
Processor: Intel® Core™ i5-4590 or AMD FX™ 8350, equivalent or better.
Memory: 4 GB RAM or more.

The cost of an equivalent system including a headset, two base stations and two controllers is in the region of 10-15,000 SEK.

3.1.2 Virtual reality platforms

One of the leading platforms for creating interactive, real-time content is Unity[23]. Unity allows for creation of 2D, 3D and VR applications and games. Unity support a wide range of VR application plug-ins where functionality is added to the platform. By creating projects, the user is able to work and interact with the content. In 2021, Volvo Cars released a project using the Unity platform [24]. The Unity AutoShowroom Template, shown in Figure 3.5, allows the user to explore and interact with a 3D model of a vehicle with basic functionality, such as opening and closing of doors. The interior and exterior views of the car, as well as the material selections in the showroom is of high detail to ensure an authentic visualization of the vehicle.

Figure 3.5: Unity AutoShowroom Template of a Volvo XC40 Recharge P8 AWD.

The AutoShowroom Template consists of multiple *GameObjects*. In Unity, GameObjects are the fundamental objects that represents props, characters and scenery [25]. They do not add any functionality themselves, but act as containers for *Components* that implement a desired function. In Figure 3.6, the GameObject is represented by the colorful coordinate axes.

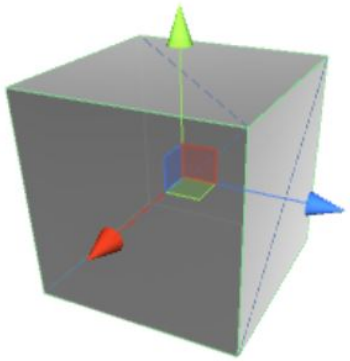


Figure 3.6: A GameObject placed inside a cube [26].

The position and orientation is always known for a GameObject in Unity. It is being tracked by a *Transform* component, as can be seen in Figure 3.7.

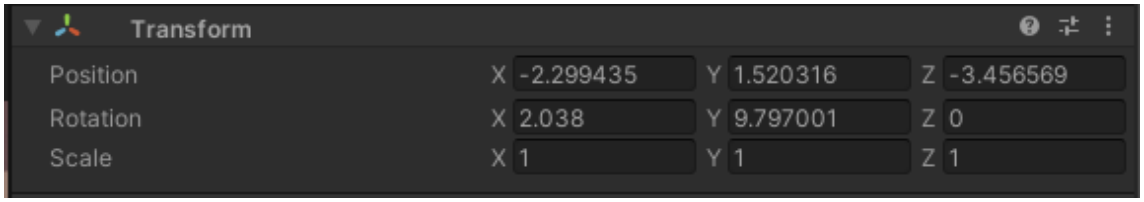


Figure 3.7: 6 DoF positioning for a GameObject in Unity.

The Transform component positions the GameObject in 6 DoF in the virtual environment. By using a motion tracking system incorporating a VR environment with GameObjects, the project did not have to develop its own platform to generate and record of motion capture. As shown in the stuby by Nakamura, Lee and Ott [3] presented in Section 1.2.1, developing such a platform would have greatly extended the scope of the project due to its complexity. The VR platform also enables use of equipment for high accuracy tracking from different tracking techniques. At this point of the project, no decision had been made about whether to use optical infrared tracking or IMUs in the final solution. With the decision to utilize VR as the platform came the opportunity to test both technologies before making a decision.

3.1.3 Robotics

Robotics can be defined as the design, construction, and use of machines (robots) to perform tasks initially performed by human beings [27]. Robots are widely used in industries where repetitive tasks are performed, often with the intention of successively replacing the human labor. Depending on the application, many companies offer a large lineup of products that are able to perform a range of different tasks. Some eminent companies within robotics are for example Boston Dynamics [28] and ABB Robotics [29].

3.1.3.1 Robot configuration

A robot mechanism, is a system of rigid bodies connected by joints [30]. The solid bodies of the robot are also known as links. Depending on the configuration, the amount of joints and links that are connected will vary. As mentioned in 3.1.1, a defined object in space has 6 DoF. DoF is also used when describing the way in which a robot can move. The total number of independent displacement or aspects of motion is defined as the degrees of freedom for the robot [31]. On the current market, there are robots with 4, 6 and 7 DoF [32]. For this project, only robots with 6 and 7 DoF were considered. A robot with 4 DoF would be insufficient. This is because it is not able to maintain a fixed orientation of axis 6, shown in Figure 3.8, when moving in three dimensions. A rotation of axis 6 will follow the rotation of axis 1 when moving sideways. The objective of the robot in this project was to reproduce a human kick and a robot with 4 DoF is too limited in its degrees of freedom for representative kicking motion.

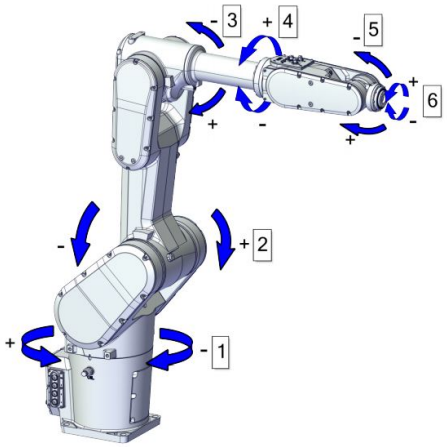


Figure 3.8: Model of IRB1300, a 6 DoF articulated robot from ABB [33]

3.1.3.2 Robot programming

There are different methods by which a robot can be programmed to perform movement. Two general areas of the methods include online and offline programming. Conventional online programming is carried out by robot operator, guiding the robot though a desired path [34]. The operator uses a teach pendant to jog the joints of the robot and is responsible for maintaining the desired orientation and position of the robot in 6 DoF. Chosen key points by the operator are recorded and saved to the robot controller. This method is known as the lead-through method. Offline programming (OLP) instead uses computer software to create robot programs, which are later downloaded to the physical robot. Virtual 3D models are used to generate and simulate robot movements. The program can be refined and fine-tuned in the software before being downloaded to the robot. OLP offers many advantages over the online programming method [34]. One of them being that OLP is more flexible than online programming, resulting in changes to the robot program can be incorporated more quickly.

There are many OLP software to chose from. It is common for companies providing robotics solutions to offer their own program. Three examples of this are RobotStudio by ABB Robotics [35], RobotExpert by Siemens [36] and K-ROSET by Kawasaki Robotics [37]. The mentioned OLP software are optimized for robots manufactured by the same company and they contain a virtual library of their own robot models. It is thus appropriate to choose a hardware and software combination from the same company when planning the robot solution.

3.1.3.3 Robot types

Two common robot types are known as collaborative robots (cobots) and industrial robots. Collaborative robots are designed to work alongside humans. Cobost enable less experiences operators without previous training the use of robots, due to their simplicity [38]. They are usually of smaller size and are known for being easy to configure and work with. One feature unique for cobots during online programming, is the ability for operators to push the robot by hand and record a lead-through program to the robot controller.

The second type is industrial robots. There are four sub-groups of industrial robots, but the most known type is an *articulated robot*. They are classified based on the amount of axes or points of rotation they have. A 6-axis is the most common configuration and one example of such a robot can be seen in Figure 3.8. Known advantages of the articulated robots are dexterity, reach and flexibility [32].

When comparing cobots and industrial robots, the later group was more suitable for this project. Some of the main advantages for cobots, including the easy to use lead-through programming and the ability to work alongside humans would not be applicable in this project. As the trajectory for the robot movement would be based on an already recorded motion by a human, it was expected to be too inaccurate to generate the program through online lead-through programming when comparing the trajectory to the original motion. An OLP software is more suited to work with when generating the robot trajectory. Among the industrial robots, an articulated robot was the most appropriate for the intended robot solution. Using the attribute of reach as an example, it was one of the most important parameters for the robot. The reach determines the length of the trajectories the robot is able to follow without making any compromises or adding limits to the original human movement.

3.2 Requirements list

The requirements list was established based on a discussion with Volvo Cars in combination with a general understanding of the problem and the scope of the project. The list consists of demands that needs to be solved by the solution as well as wishes that should be solved to the best extent possible, but are not necessary for solving the main objective of recording and recreating human motion. The requirements targeted both the motion capture and robot solution.

Area	Criteria	Verification method	Target value	Demand/Wish
	1. Convenience			
Robot	1.1 Allow operator in close proximity during operation	Product design	Operator located in the same room	Wish
Robot	1.2 Solution moveable within the test area	Product design	Reasonable effort to relocate solution	Demand
Robot	1.3 Easy to operate with reasonable physical effort	Subjective and functional testing	Only one operator is required	Wish
	2. Functionality			
Robot	2.1 Activate FMDM sensor on a vehicle from Volvo Cars	Functional testing	Useable solution	Demand
Robot	2.2 Perform a representative kick as by a human leg from the knee down	Functional testing	Comparable kick trajectories by human and robot	Demand
Robot	2.3 Perform a kicking motion by following a trajectory from an input file	Product design	The robot motion has a strong resemblance to the input file	Demand
Robot	2.4 Perform a kicking motion in normal speed	Functional testing	The robot motion can follow a set speed	Demand
Robot	2.5 Perform a kicking motion in various speeds	Functional testing	The robot motion can vary between different speeds during motion	Wish
Robot	2.6 Allow interchangeable shoe attachments	Product design	Any shoe can be attached to the solution	Wish
Robot	2.7 Automated test programs including many kicking motions	Product design	The robot can perform many kicking motions without interupption	Wish
Motion Capture	2.8 Capture horizontal body movement	Functional testing	No limitation in the motion tracking	Demand
Motion Capture	2.9 Capture vertical body movement	Functional testing	No limitation in the motion tracking	Demand
Motion Capture	2.10 Capture angular movement from joints	Functional testing	No limitation in the motion tracking	Wish
Motion Capture	2.11 Record timelapse of body movement	Functional testing	The timelapse can be used to determine the speed of the recorded kick	Wish
Motion Capture	2.12 Body movement is expressed in a data set	Product design	Data from a run can be explicitly expressed	Demand
	3. Reliability			
Robot	3.1 Not block the movement of the tailgate after sensor activation	Functional testing	The robot does not interfere with the tailgate at any time	Demand
Motion Capture	3.2 Test setup enables repeatable outputs	Functional testing	No use of temporary equipment in test setup	Demand
Motion Capture	3.3 Test setup can be used by different body types	Functional testing	Any test person may be subject to motion capture	Wish
	4. Durability			
Robot	4.1 Repeatable positioning in relation to current test vehicle	Product design	The solution can be accurately positioned in the test environment	Demand
	5. Ergonomics			
Robot	5.1 Not make a lot of noise	Subjective and functional testing	The operator does not experience the solution as intrusive	Wish

Table 3.1: Initial requirements list

The robot will be stand-alone from any vehicle it interacts with and thus only 2 out of 20 requirements included any criteria with a car. First, it is performing the main function, *2.1 Activate FMDM sensor on a vehicle from Volvo Cars*. Second, is the adaptation to the car's response of the sensor activation, *3.1 Not block the movement of the tailgate after sensor activation*.

In order to evaluate which motion tracking equipment and robot model that was suitable to use in the project, presented in the next Section 3.3, the requirements list acted as a basis for the decisions.

3.3 Procurement of motion tracking equipment and robot

Presented below is the equipment the project used during the initial testing activities.

3.3.1 Motion tracking equipment

The department known as *Open Innovation Arena* at Volvo Cars was contacted in order to get access VR equipment and software. The project was allowed to use licenses enabling work with a VR environment in Unity and got access to various high-end equipment used for the initial testing of a motion tracking solution. The specifications of the computer being used was:

Graphics: NVIDIA® GeForce® RTX 2080 TI.
Processor: Intel® Core™ i9-9900K.
Memory: 32 GB RAM.

Equipment using both optical infrared motion capture and IMUs became available. The setup with a headset, a tracker and two base stations, shown in Figure 3.9, enabled optical infrared tracking.

(a) Valve Index Base Station [39]

(b) HTC Vive Tracker [40]

(c) Varjo VR-1 Headset [41]

Figure 3.9: Equipment enabling optical infrared motion capture

The equipment to perform motion tracking using IMUs was the Teslasuit. The suit can be seen in Figure 3.10. The suit includes ten integrated IMUs, where six are placed in the jacket and four are placed in the pants. The suit allows integrated skeletal and 3D kinematic motion capture in the virtual environment, which is visualized by a full body avatar [42]. As there are no IMUs attached to the feet of the user wearing the suit, its positions are estimated based on the pose of the virtual avatar.

Figure 3.10: Teslasuit [21].

The available equipment using both optical infrared motion capture and IMUs fulfill the demands of the motion capture presented in the requirement list, Table 3.1. The demands of the solution includes capturing both vertical and horizontal movement, criteria 2.8 and 2.9, which is not an issue when using either of the methods. It also covers the requirement of repeatable outputs, criteria 3.2. As the same equipment will be used and the test setup can be recreated for different motion recording sessions, this criteria is also fulfilled. The last demand of the motion capture system is the ability to express the body movement in a data set, criteria 2.12, as this will be used as input for the robot movement. When utilizing the VR platform Unity for both solutions, the captured motion is digitized and thus becomes available to export as a data set when using a functional motion recording method.

3.3.2 Robot

The company ABB Robotics was contacted in order to get access to an articulated robot. Two different aspects of this decision was that ABB offers their own offline robot programming software RobotStudio [35], as well as having a large library of different robots that would fulfill the demands for the robot in the requirements list, presented in Table 3.1. The robot acquired by the project was the model IRB1200, shown in Figure 3.11.

Figure 3.11: Robot model IRB1200 from ABB.

The full specification of the robot can be found in Appendix A.1. Key specifications of the model that was important to the project are the following:

1. 6-axis
2. 0.9 m reach
3. 52 kg

In the requirements list in Table 3.1, there are four demands that are possible to fulfill by using a 6-axis articulated robot. These criteria are 2.1 activate the FMDM sensor, 2.2 perform a representative kick as by a human from the knee down, 2.4 perform a kicking motion in normal speed and 3.1 not block the movement of the tailgate after sensor activation. The robot IRB1200 can perform motions in various speeds and has enough axles to represent a human kick from the knee down. Criteria 3.1 can be fulfilled by continuing a motion away from the tailgate after the FMDM sensor activation, as the tailgate is not opened instantaneously. The weight of 52 kg is subjectively not considered heavy enough to restrict moveability within a test area, as criteria 1.2 states. Because the robot is moveable, it is also possible to identically position the robot during different test session and for different test vehicles. As criteria 4.1 states, repeatable positioning is thus achievable. Criteria 2.3, to perform a kicking motion by following a trajectory from an input file, is possible by implementing a suitable method with the OLP software RobotStudio. The program has a virtual model of the IRB1200 in its library, which can generate robot programs that can be downloaded to the physical robot. The specification of 0.9 m reach is referring to the distance from the center of the robot to the outer edge of the robotic arm when in a fully extended position. 0.9 m is subjectively not considered to be a significantly limiting factor when assessing the length of the trajectories the robot can perform.

3.4 Initial testing of chosen solutions

The tests were designed to have outcomes that would indicate if the equipment was suitable to incorporate in the final solution of the project. For the motion tracking, the test included both tracking techniques and their functionality and reliability were compared relative to each other. In order to generate a recording with Cartesian data output from a motion capture, a script was created in Unity, written in the programming language C#. The generated Cartesian data output file was post-processed to enable visualization of the data. The post-processing was performed in a Google Colaboratory Notebook and is written in the programming language Python. The robot movement was simulated in ABB Robotics' software RobotStudio. The test was designed to generate robot trajectories based on coordinate input.

3.4.1 Human motion tracking

A project in Unity was created and used as the initial test environment. The project included a model of an XC40 Recharge P8 AWD and an avatar representing the Teslasuit [21]. The avatar was positioned behind the car. The figures below showcase the opened and closed state of a successful tailgate opening sequence. The project was created to increase the understanding of the use case of being placed behind a car and performing kicks open the tailgate.

(a) Closed tailgate. (b) Opened tailgate.

Figure 3.12: XC40 Recharge P8 AWD 3D Unity template with a Teslasuit avatar.

To focus on the visualization of the Teslasuit avatar during the motion capture, a simplified project containing only an avatar and a GameObject inside a cube, representing a HTC Vive Tracker, was created. A GameObject was placed on the right leg of the avatar, which followed the movement of the leg. The Teslasuit and the tracker were independent from each other in the model, but the tracker was graphically positioned on the right shoe of the avatar. This means that the position in the *Transform* component for each GameObject only changed because of real movement of the physical equipment and not due to graphical changes of the other GameObject on the Teslasuit avatar. The project is shown in Figure 3.13, where the colorful coordinate axes represent the GameObject connected to the Teslasuit and the orange cube on the right foot of the avatar is representing the GameObject

connected to the HTV Vive Tracker.

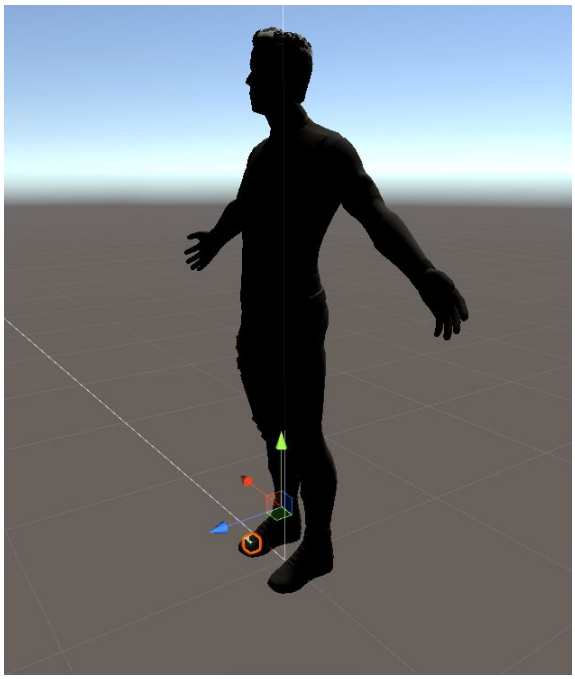
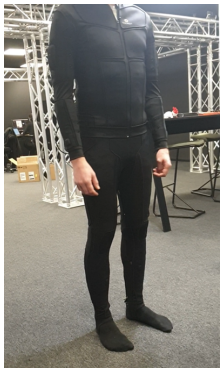


Figure 3.13: Project with a Teslasuit avatar and a GameObject representing a HTC Vive Tracker.

Figure 3.14 shows the worn equipment during the initial motion tracking testing. As there was no benefit for the user to visualize the project shown in Figure 3.13 in VR, the Varjo VR-1 headset was not worn. However, in order to generate a VR environment where the motion of the HTC Vive Tracker can be tracked, it needs to be connected to the project.



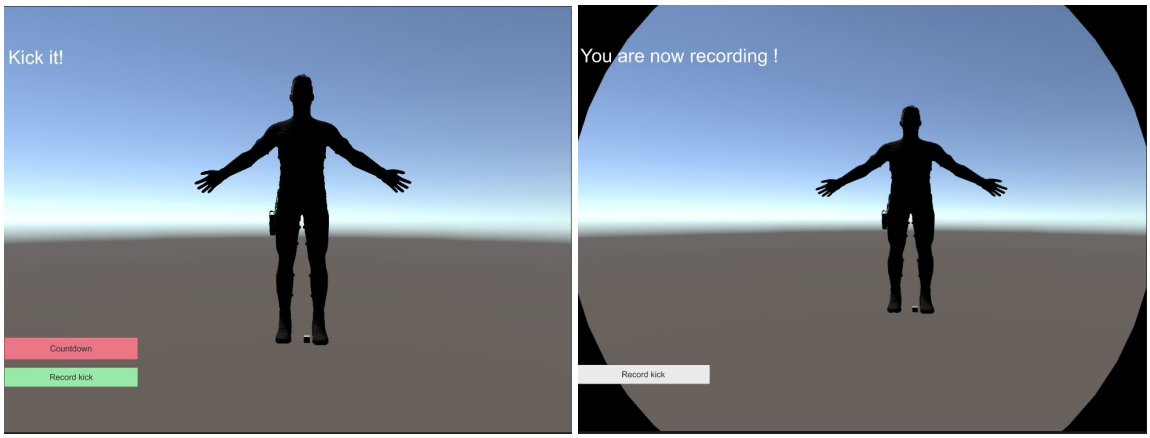
(a) Project member wearing the Teslasuit.

(b) HTC Vive Tracker placed on shoe.

(c) Valve Index Base Station placed on stand.

Figure 3.14: Used equipment during initial motion tracking testing.

The positioning is updated in real time as the GameObjects move. When connected to the VR environment, the positions of the GameObjects are based on the corresponding movements of the physical objects, i.e. the Teslasuit and the HTC Vive Tracker. In order to enable recording of the data from the motion and perform it during controlled conditions, a graphical user interface (GUI) was created in Unity. The GUI's main function was to start and stop the data recording. The recording was initiated as the user pressed the green button saying "Record kick" in Figure 3.15a.



(a) Interface to initiate recording. (b) Interface during recording.

Figure 3.15: GUI for recording the positions of the GameObjects.

The complete scripts used for recording and extracting the coordinates from the *Transform* component of the GameObjects placed on the leg of the Teslasuit avatar and the HTC Vive Tracker is shown in Appendix B.1 and B.2, respectively. The logic for how each script records the position of the GameObject during the movement is the following:

1. Wait for user to click the button "Record" to initiate the recording
2. Create a text file for the data to be stored
3. Save and write the initial x,y,z-coordinates to the text file
4. Write updated position to the text file until the recording is stopped

Shown below is the part of the code being updated once per frame that handles the updating of the x,y,z-coordinates and prints the values to the text files *LegPosition* and *TrackerPosition* previously created in the script. The value of the recorded coordinates is based on the relative movement of the GameObject from its initial position. This is expressed as *initPosx - transform.position.x;*, where *initPosx* is the saved position from when the recording started and *transform.position.x* is the current position of the GameObject in the virtual environment.

```
// Update is called once per frame

void Update()
{

 writing = GameObject.Find("Record").
 GetComponent<ClickRecord>().writing;

 x = initPosx - transform.position.x;
 y = initPosy - transform.position.y;
 z = initPosz - transform.position.z;

 string xs = string.Format("{0:0.0000}", x);
 string ys = string.Format("{0:0.0000}", y);
 string zs = string.Format("{0:0.0000}", z);

 if (writing)
 {
  srPos.WriteLine(xs.ToString() + "," + ys.ToString() +
  "," + zs.ToString() + ",");
 }
}
```

The generated text files *LegPosition* and *TrackerPosition* consisted of all the recorded positions of the GameObjects during the recording. In order to verify that the data recording process was successful, a Python script was created in a Google Colaboratory Notebook. The full script can be found in Appendix C.1 but a simplified flow diagram of the code is shown in Figure 3.16 below.

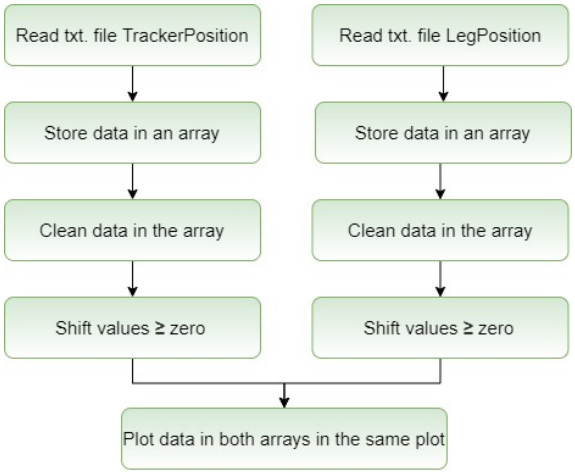


Figure 3.16: Flow diagram for the Google Colaboratory Notebook during initial motion tracking testing.

The script used *LegPosition* and *TrackerPosition* as inputs and visualized the data by creating a plot of the two trajectories. The plot included position and direction of travel. The plot can be found in Section 4.1.1, *Human motion tracking*. The data in the text files was post-processed to ease the visualization and increase the usability. The post-processing removed adjacent identical coordinates, as this implied that the GameObject had not moved since the last frame. The reason for removing such coordinates is that they do not add any information to the captured trajectory of the motion of the GameObjects, as means that the Teslasuit and the HTC Vive Tracker was not moving during the interval of when these frames were recorded. The processing also included shifting all values to be equal or greater than zero. This was in order to ease the process of working with the coordinates in any 3D modeling software. As described in Section, 2.3, CATIA V5 requires positive values for the coordinate import via Excel. The logic of how the values were shifted to values equal or greater than zero is shown below. The Cartesian coordinates in the two text files are represented by *x1*, *y1* and *z1* and *x2*, *y2* and *z2*, respectively.

```
minx=min(min(x1),min(x2))
miny=min(min(y1),min(y2))
minz=min(min(z1),min(z2))

if(minx<0):
  x1=x1+abs(minx)*np.ones_like(x1)
  x2=x2+abs(minx)*np.ones_like(x2)
if(miny<0):
  y1=y1+abs(miny)*np.ones_like(y1)
  y2=y2+abs(miny)*np.ones_like(y2)
if(minz<0):
  z1=z1+abs(minz)*np.ones_like(z1)
  z2=z2+abs(minz)*np.ones_like(z2)
```

3.4.2 Mechanical movement

A project in RobotStudio was created and used as the initial test environment. The project included a model of a Volvo V60 [43], four wheels [44], a shoe [45] and the ABB robot IRB1300, imported from the RobotStudio's default library. At the time of performing this test, it was not yet decided which model of the articulated robot from ABB Robotics that would be used. The acquired robot by the project was of the model IRB1200, as described in 3.3.2. The RobotStudio project is shown in Figure 3.17 below. The robot was positioned behind the car to mimic a realistic scenario where the robot is used to open the tailgate.

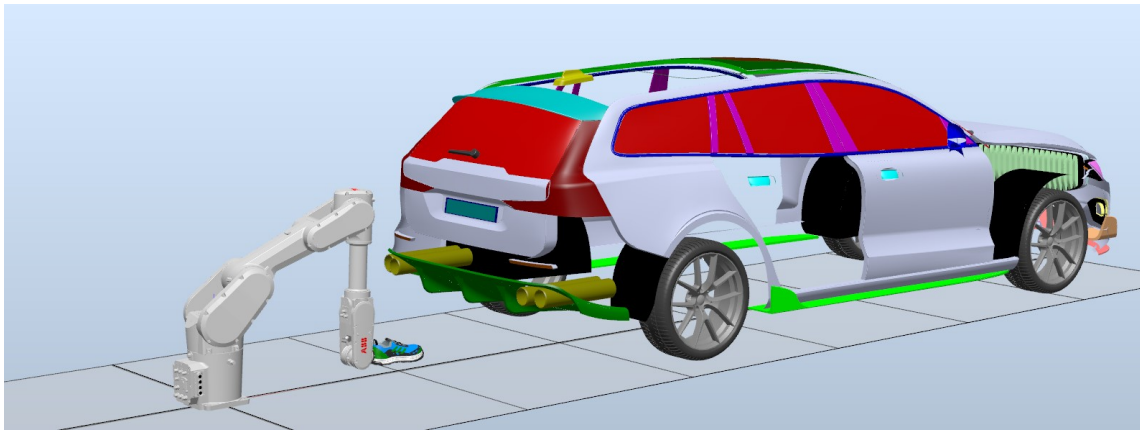


Figure 3.17: RobotStudio project with models of a car, four wheels, a shoe and a robot.

In order to perform a kicking motion with the robot, a number of *robtargets* had to be created in the model. A trajectory can be created by selecting multiple *robtargets* and adding them to a path. The robot can move along the trajectory when a path is created. A flow diagram of the logic of generating robot movement by manually adding *robtargets* is shown in Figure 3.18.

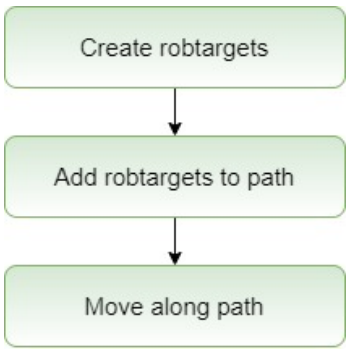


Figure 3.18: Flow diagram of how to generate robot movement by manually adding *robtargets*.

The instruction to move between every pair of coordinates in the path, together forming the full trajectory, is generated by a *move* instruction in RobotStudio. The robot will pass through all positions defined by the *robtargets* when moving along the path. The *robtargets* were manually added as shown in Figure 3.19.

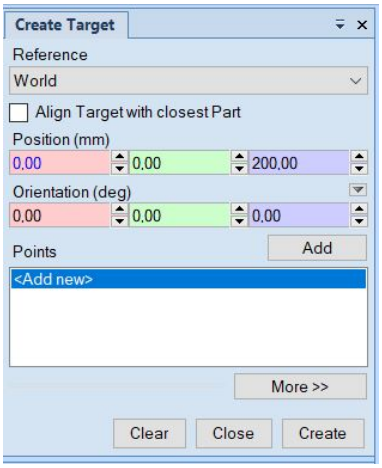


Figure 3.19: Window for manually adding *robtargets*.

The resulting trajectory in RobotStudio from manually adding *robtargets* and creating a path is shown in Section 4.1.2.

3.5 Adaptation of solution for motion tracking related to concept limitations

At the time of writing this report, RobotStudio does not have a function to enable speed data to be a parameter imported by the user. No further efforts were therefore made during the motion recording to capture the speed of the movement, as this would not be usable information when generating robot programs. RobotStudio is also limited in such a way that the robot is only able to follow one trajectory in the simulation. Capturing the movement of both the Teslasuit and the HTC Vive Tracker was due to this redundant, as only one data set of coordinates would be able to be used in the final robot program. The decision of which tracking method to use was based on the physical limitations of the equipment. As already mentioned in Section 3.3.1, the used Teslasuit is of a ten IMU sensor configuration. The full body avatar is generated in the VR environment by estimating the position of the body parts that are not covered by real position data captured from one of the IMUs. Figure 3.20 shows the placement of the sensors that can be seen from the front view when wearing the Teslasuit.

Figure 3.20: Front view of sensor placement for a Teslasuit with 10 IMU sensors [42].

All four IMUs placed inside the pants are shown in the figure. Due to that the two lowest sensors are placed on the lower leg, and not on the feet, the position of the feet are estimated based on the position of the legs. This results in the feet being a non-dynamic part of the Teslasuit avatar in the VR environment. If a GameObject is placed on the feet of the avatar, its position would not change if a person wearing the suit would only move its wrist and not the entire leg with it. A human performing a kick to open the tailgate of a vehicle from Volvo Cars is expected to perform movements with the feet that are not strictly limited by the position of the leg. Using the Teslasuit avatar would exclude such movement from the recorded data. By placing the HTC Vive Tracker on the foot, as shown in Figure 3.14b, the movement of the wrist itself is captured and thus included in the motion recording. When comparing the data of the motion recording from the initial testing performed in Section 3.4.1, the quality and accuracy are of equally high standard. The plot with the captured trajectories being recorded with the Teslasuit and the HTC Vive Tracker, is visualized in Figure 4.1. As there is no difference in the quality of the recorded data, performing motion tracking with the HTC Vive Tracker is more suitable for this project as it captures the real position of the wrist.

The robot the project had access to, one of model IRB1200, has a 6 DoF configuration. This limits the degree by how much the the robot movement can mimic the human movement. Neither of the implemented methods of importing Cartesian data to RobotStudio, presented in Section 3.6.2, allows the inclusion of angular position of the foot during the trajectory. Due to this limitation in RobotStudio, no further efforts were made to record the rotation and thus the angular movement of the GameObjects during the motion capture.

3.6 Detailed prototype design and construction

The design of the final prototype included additional development and refinements to both the motion tracking technique and robot movement. Efforts were also made to find a suitable interface between the export of the data files from the Google Colaboratory Notebook and the import of the coordinates to the RobotStudio. The ambition was to streamline the process of going from human to corresponding robot motion with as few required steps in between as possible. Figure 3.21 shows the desired process.

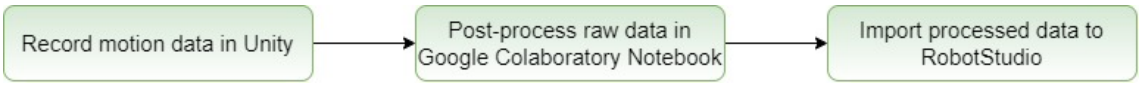


Figure 3.21: Flow diagram of desired intermediate steps to generate robot movement.

3.6.1 Human motion tracking

The same project in Unity, as shown in Figure 3.13, was used for further recordings. However, as described in 3.5, only the HTC Vive Tracker was connected to the VR environment. A test setup including a Volvo XC90, shown in Figure 3.22 below, was used to enable data recordings in a realistic user scenario where motion underneath a car's bumper is performed. The same setup of equipment used for optical infrared motion capture during the initial testing, presented in Section 3.4.1, was used for this testing. Multiple kicks were performed and saved as individual data files. The test proved that the motion tracking system could manage different types of kicks and generate accurate motion data regardless of the movement.

Figure 3.22: Placement of equipment during motion capture recordings.

Some changes and adaptations were made to the Google Colaboratory Notebook used to process the output text files compared to the one used during the initial testing, as described in Section 3.4.1. First, there was just one trajectory being plotted. As the HTC Vive Tracker was the only connected equipment enabling motion capture, it was the movement of the tracker being visualized. Second, changes to the output format of the data processing from the raw text file was made. The changes made the data align with the preferred format in RobotStudio. This included:

1. Cleaning the data file. This was performed with the same logic as used previously, by removing adjacent identical coordinates.

2. The values in the text file were multiplied with 1000, in order to be converted to millimeters.

3. The sign for the y-values was shifted, making negative values positive and positive values negative. RobotStudio employs a right-handed coordinate system while Unity uses a left-handed system. In order to not generate a mirrored robot trajectory compared to the recorded human movement, the y-vales had the be shifted.

4. Generate a CSV-file of the post-processed data. This was the file format used for the RobotStudio add-in described in Section 3.6.2.2.

Four kicks recorded during this session are shown in Figure 4.3. The trajectories are plotted after the data processing. The complete Google Colaboratory Notebook is presented in Appendix C.2.

3.6.2 Mechanical movement

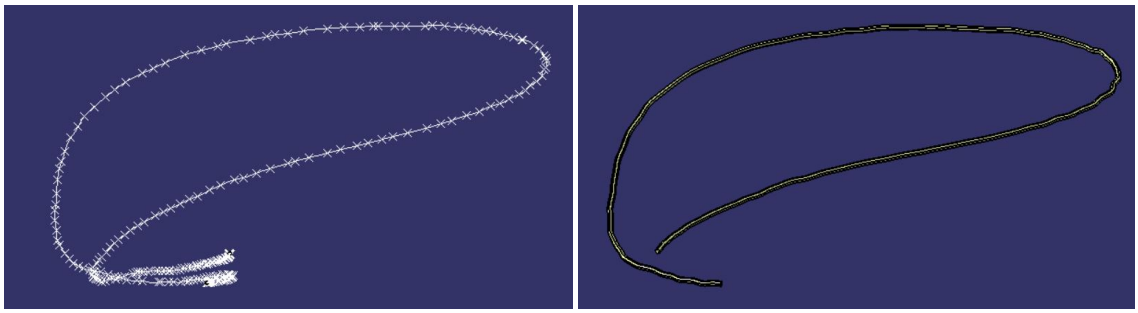
Based on the data file of coordinates created as output from a motion recording, efforts were made to import the list of coordinates as a trajectory to RobotStudio.

3.6.2.1 CATIA V5

Initially, CATIA V5 was used as an intermediate software. Using the built-in function described in 2.3, data from the coordinate file could be imported as a point cloud with a spline connecting the points. The import was performed by adding the coordinates in the data file to the Excel sheet *GSD_PointSplineLoftFromExcel*.

Two planes were created in the model, one perpendicular to the first coordinate and one to the last. A sketch with the profile of a hexagon was added to the first plane, as can be seen in Appendix E.1. By using the action *Sweep* along the spline between the planes, a solid geometry was created. Multiple manual editing iterations of the imported coordinates from *GSD_PointSplineLoftFromExcel* were necessary for the solid geometry to be created. The point cloud after the editing, used to create the solid geometry, is shown in Figure 3.23a. The manual editing mainly

involved reducing the amount of coordinates forming the spline. The high density of coordinates in the initial import in combination with the spline almost passing through the same coordinates during the beginning and end of the recorded motion capture was the reasons for the difficulties to create an approved solid geometry in CATIA V5. The solid geometry, shown in Figure 3.23b, was exported as a STEP-file.



(a) Imported point cloud and spline. (b) Created solid geometry.

Figure 3.23: Models in CATIA V5 based on imported coordinates.

The STEP-file was imported the the model shown in Figure 3.17. By using the built-in function in RobotStudio called *AutoPath: "Create a path from the edges of a geometry or a curve*, a trajectory for the robot in the model was created. The imported geometry had to be manually positioned in the robot station to start at a suitable position.

3.6.2.2 RobotStudio add-in

Due to the encountered issues with the solid geometry creation in CATIA V5, a discussion with ABB Robotics about alternative solutions was initiated. A description of the desired import process, with no intermediate software between Unity and RobotStudio and minimal user actions required, lead to ABB sharing an add-in not available as a default download for RobotStudio. The add-in is named *Coordinate File Import*. The add-in had to be refined and customized to function with the coordinate text files generated from the motion recordings. It allows for import of coordinates, based on a comma separated values (CSV) file, and generates a trajectory passing through all coordinates. The add-in also has the functionality to remove coordinates within in minimal set distance of each other. The generated geometry had to be manually positioned in the robot station to start at a suitable position.

3.6.2.3 Robot appearance

A creative process started when making a realistic appearance of the robot compared to a human leg. Initially, a solid leg was considered as it had the correct look of a human leg. It became clear that such a design was not useful when simulating a leg in motion. The angle between the foot and the leg would always be approximately

90 degrees. The angle between the foot and the leg changes during the process of performing a kicking motion underneath a bumper. When a person is standing in a straight pose before initiating a kick, the appearance of the leg in Figure 3.24 is correct.

Figure 3.24: 3D printed leg.

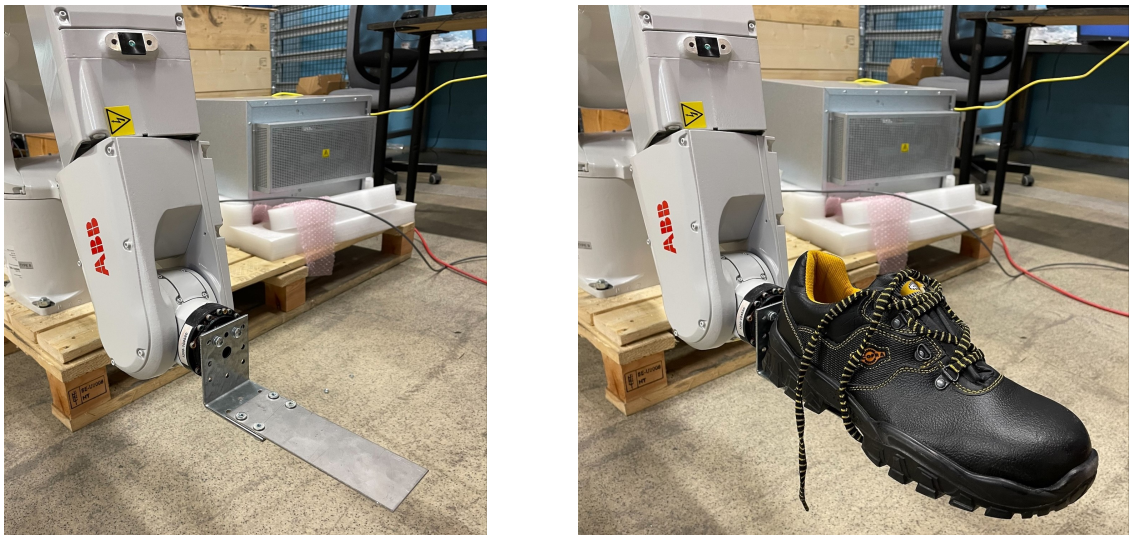
However, when the foot is underneath the bumper in an extended position, the angle between the foot and the leg is widened. In order to design the robot appearance to allow the angle between the foot and leg to change, either a joint needs to be introduced between two solid parts inside the shoe or a flexible material that follows the extension and retraction of the robot arm during a kick should be used. A solution using flexible materials is shown in Figure 3.25.

(a) Side view of an extended position. (b) Top view of an extended position.

Figure 3.25: Solution to replicate a human leg at an extended kicking position.

In order to recreate the shape of a leg, a cylinder was placed inside a plastic bag. The plastic bag was attached to a bracket mounted on the robot. The plastic bag was replaced with jeans material for the final prototype. The Figure 3.25a shows the similarities of the profiles for a human lower leg and the prototype in an extended

kicking position. Figure 3.26 shows the support structure to place the shoe on the robot. The shoe was later attached to the robot by using a strap or tape. The solution did not impose any restrictions on which shoe that could be used in the final prototype. The final setup for the human leg replication is shown in Figure 4.6.



(a) Attached bracket to the robot.

(b) A shoe placed on the bracket.

Figure 3.26: Solution to place a shoe on the robot.

3.7 Verification of solution

The functionality of the final prototype was verified by testing with a real vehicle. The robot, bolted to a EU pallet, was placed behind a Volvo XC60. The gray box underneath the robot has the same dimensions as the EU pallet and was included to ease the positioning of the trajectory in the model so the physical robot would not collide with the pallet. The tip of the shoe was 35 cm behind the vertical edge of the car's bumper. The trajectory used for the testing can be seen in Figure 3.27. The trajectory was imported to the robot program by using the add-in described in Section 3.6.2.2. The data in the trajectory was recorded when performing a straight kick and the complete data set of coordinates can be found in Appendix F. The full code generated in the robot station to perform the testing is presented in Appendix D.2.

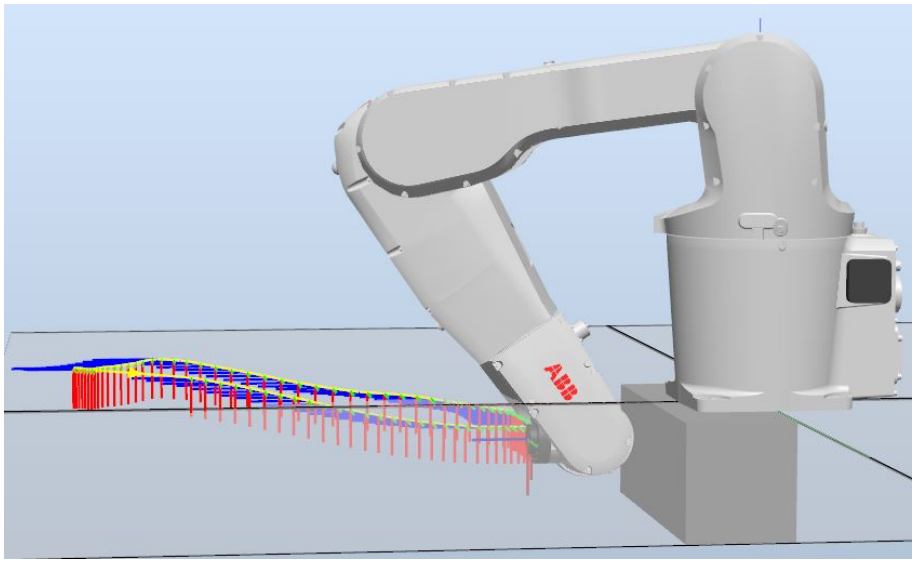


Figure 3.27: Trajectory used for verification testing shown in RobotStudio.

Figure 3.28a shows the robot in the initial position of the trajectory. Figure 3.28b shows the maximum extended position of the robot. It was in the extended position that the activation of the FMDM sensor was desired.

(a) Robot in home position. (b) Robot in extended position.

Figure 3.28: Side view of robot positions during verification test.

In order to verify requirement 2.1 *Activate FMDM sensor on a vehicle from Volvo Cars*, as presented in Table 3.1, a quantitative test was designed. The result of the test would show if the robot could repetitively trigger the FMDM sensor by performing a kicking motion. The test program performed 100 consecutive identical kicks, with a six second wait time between each kick. The speed of the robot movement was set to 800 mm/s. The code used to run the test program is shown below.

```
PROC main()
    mySpeed.v_tcp:=800;
    FOR i FROM 1 TO 100 DO
        TPWrite "Kick: "+valtostr(i);
        Straight_1;
        WaitTime 6;
    ENDFOR
ENDPROC
```

While performing the test, the amount of kicks that had been performed was printed on the robot pendant, as shown in Figure 3.29. The test program of 100 kicks was ran three times and each session had the duration of 12 minutes and 30 seconds.

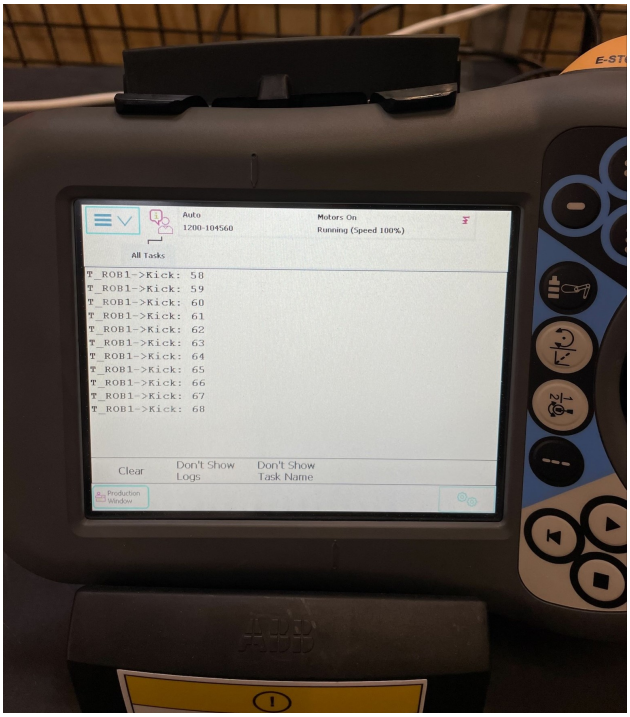


Figure 3.29: Window on robot pendant during test program.

Neither the speed or the trajectory was changed during the 300 kicks. In order to verify the functionality of repetitively being able to activate the FMDM sensor, only identical movement from the robot could be used. It is the only way to determine whether potential variations in the results depends on the motion input or the FMDM sensor itself. If identical input results in varying results for activation of the sensor, it constitutes the basis for the need of future development of the unit. Such development is outside the scope of the project but the final prototype is suitable to be used in the process of verifying progress in the motion recognition of kicks performed by the FMDM sensor.

4
Results

This chapter presents obtained results from the different development activities, presented in Chapter 3.

4.1 Initial testing of chosen techniques

Below is a description of the results from the initial tests performed with the chosen techniques for the human motion tracking and robot movement.

4.1.1 Human motion tracking

By using the Google Colaboratory Notebook presented in Appendix C.1, a visualization of the data for the movement of two GameObjects during a recording was created. One GameObject was based on the position of the Teslasuit and the other based on the position of the HTC Vive Tracker. The physical setup for the motion recording is shown in Figure 3.14. The movements in three dimensions and the direction of travel, indicated by the blue arrows, are presented in the plot. The data in Figure 4.1 below is visualized after the processing in the Google Colaboratory Notebook. A flow diagram for the code in the notebook is shown in Figure 3.16.

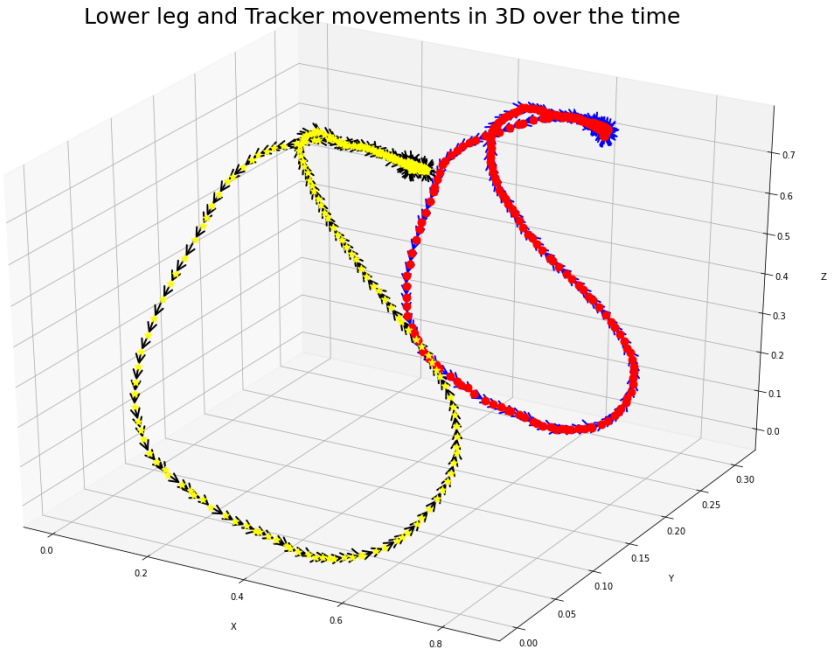


Figure 4.1: Recorded data for movement of the Teslasuit GameObject, shown in red, and the HTC Vive Tracker GameObject, shown in yellow.

The reason why the trajectory of the HTC Vive tracker is longer compared to the one of the Teslasuit, is because of the placement of the GameObjects that generates the motion data. When performing a forward kick from an initial straight position, the length of travel for the foot is longer than it is for the lower leg. This is shown by the yellow trajectory of the HTC Vive Tracker, placed on the shoe of the kicker, is longer than the trajectory of the Teslasuit, recorded from the lower leg.

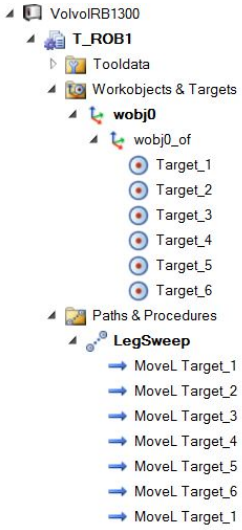
4.1.2 Mechanical movement

Resulting trajectory in RobotStudio after manually adding six *robtargets* to the robot station and creating a path to generate *move* instructions.

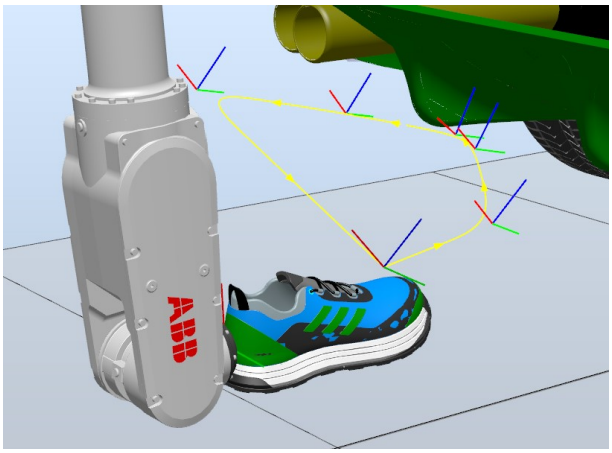
The full code generated in the robot station is presented in Appendix D.1. The first *robtarget* and *move* instruction in the robot station are the following:

```
CONST robtarget Target_1:=[[1064.93497425,-1.581796012,
    214.200960295],[0.011884697,-0.398162114,0.02186256,
    -0.916977488],[-1,0,2,1],[9E+09,9E+09,9E+09,9E+09,
    9E+09,9E+09]];

 MoveL Target_1,v800,z100,MyTool\WObj:=wobj0;
```



(a) Created *robtargets* and *move* instructions



(b) Trajectory in RobotStudio.

Figure 4.2: Robot station with a trajectory and corresponding *robtargets* and *move* instructions.

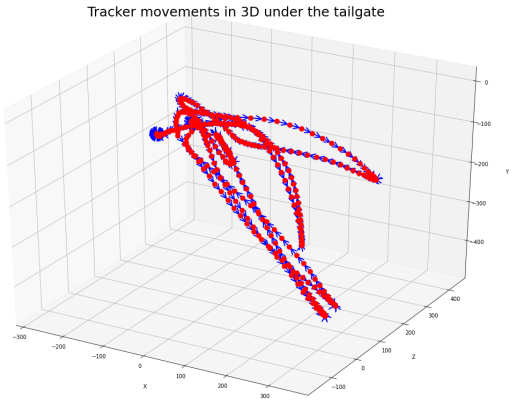
4.2 Detailed prototype design and construction

The following sections cover the results from the detailed design and construction of the final prototype.

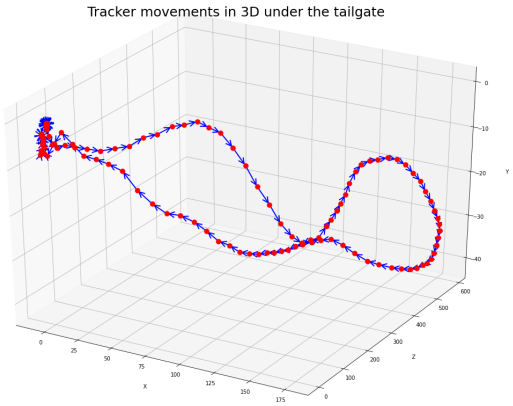
4.2.1 Motion tracking

By using the Google Colaboratory Notebook presented in Appendix C.2, a visualization of the data for the movement of a GameObject during a recording was created. The GameObject was based on the position of the HTC Vive Tracker, placed on the shoe of the kicker, as shown in Figure 3.14b. The movement in three dimensions and the direction of travel, presented by the blue arrows, is presented in each plot. The figures visualize the movement of four separate recordings of a human kick underneath the bumper of a Volvo XC90. The test setup for the recordings is shown in Figure 3.22. The data in Figure 4.3 below is visualized after the processing in the Google Colaboratory Notebook. A description of the code in the Google Colaboratory Notebook is shown in Section 3.6.1.

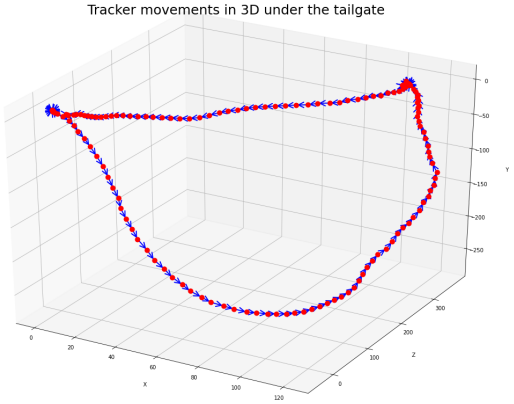
By visualizing the data from the recordings, it is possible to assess the quality of the motion capture. Each of the four plots in Figure 4.3 show complete data sampling throughout the full trajectory. The result proves that the motion tracking method is able to successfully capture and record different types of kicks under the bumper of a Volvo XC90.



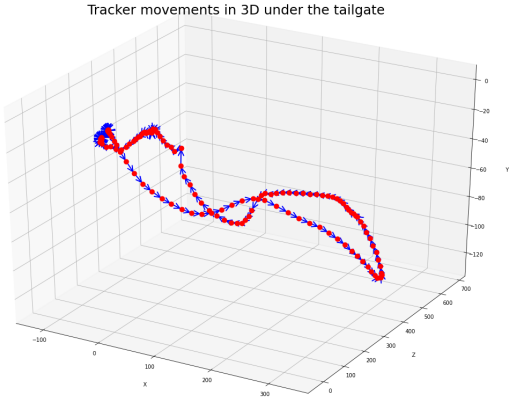
(a) Visualization of recorded kick 1.



(b) Visualization of recorded kick 2.



(c) Visualization of recorded kick 3.



(d) Visualization of recorded kick 4.

Figure 4.3: Visualization of motion capture data from the HTC Vive Tracker for four human kicks triggering the FMDM sensor underneath the bumper of a Volvo XC90.

4.2.2 Mechanical movement

The Figure 4.4 below shows the generated trajectory in RobotStudio, based on the imported solid geometry from in CATIA V5, when using the build-in Function *AutoPath*.

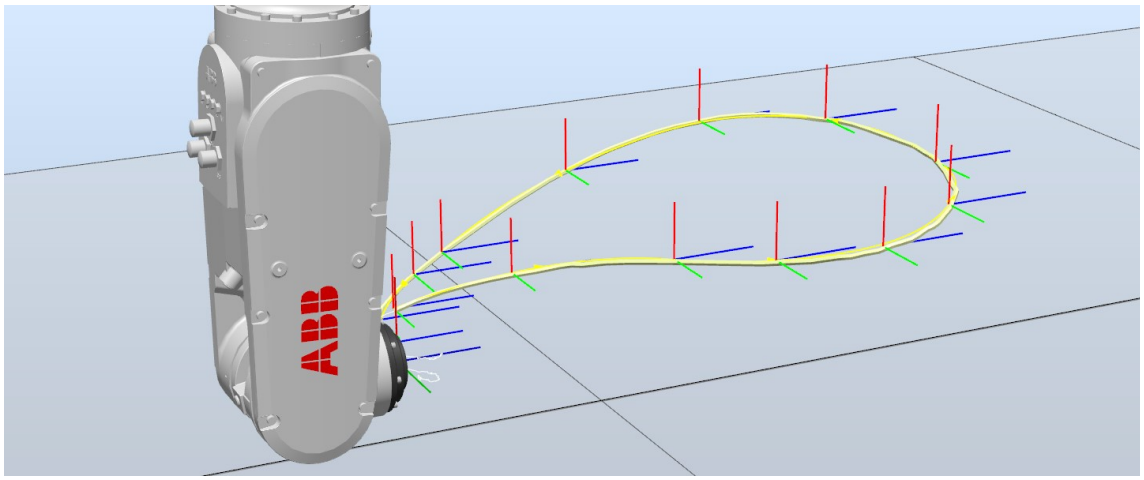


Figure 4.4: Trajectory generated from imported geometry from CATIA V5.

The Figure 4.5 below shows the generated trajectory in RobotStudio when using the add-in *Coordinate File Import*.

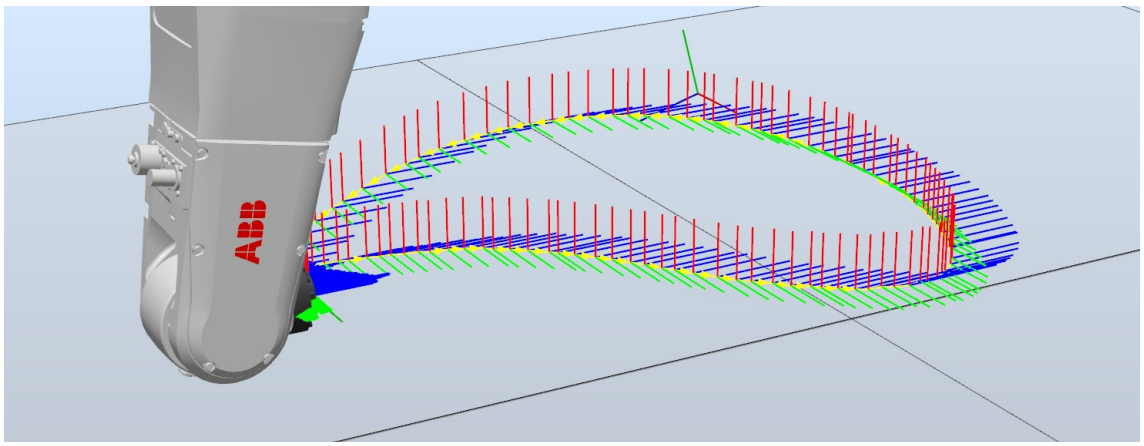


Figure 4.5: Trajectory generated from using RobotStudio add-in.

The Figure 4.6 below shows the final setup of how the appearance of a human leg is replicated when located in an extended kicking position. The jeans bag is put inside any shoe placed on the bracket. The shoe is attached to the bracket with tape.

Figure 4.6: Final setup for human leg replication.

4.3 Verification of solution

The Table 4.1 below shows the success rate for the FMDM sensor activation, for the three test programs that was performed. A kick triggering the FMDM sensor was defined as a kick activating the full sequence of triggering the FMDM sensor leading to a successful opening or closing of the tailgate. The robot program described in Section 3.7, was used to perform identical straight kicks with the same velocity with the robot.

	Test run 1	**Test run 2**	**Test run 3**
Kicks triggering the FMDM sensor	**88**	**83**	**78**
Performed kicks	**100**	**100**	**100**
Success rate (%)	**88**	**83**	**78**

Table 4.1: Verification test program results

5
Discussion

This chapters carries a discussion about the areas that are considered relevant for an additional understanding of the project.

5.1 Fulfillment of requirements list for final prototype

The requirements list, shown in Table 3.1, was the basis when deciding which techniques and equipment would have the potential to solve the main objectives. The discussion of why the chosen equipment could be combined to meet the demands in the requirements list is held in Section 3.3 *Procurement of motion tracking equipment and robot.*

The requirements list consists of five main areas that the final prototype should address. The first one, *Convenience*, targets the robot solution. By choosing a robot weighing 52 kg, as can be seen in the robot specification in Section 3.3.2, and placing it in a EU pallet during the verification testing, shown in Figure 3.28, the robot solution was moveable and able to be operated with reasonable physical effort. During the testing with the robot, it experienced some shaking when performing movements. This was due the robot being bolted to an EU pallet and not something heavier and more rigid, most preferably the floor itself. As the priority for the project was to have the robot movable to allow for easy positioning in relation to the car used for testing as well clearing for external work activities being performed at the same test area, the shaking was considered acceptable. Operators are allowed to be in close proximity to the robot while it is performing a kick, as long as the robot is allowed to move freely without risk of collision within its range of operation. The model being used, IRB1200, has a range of 0.9 meters, as presented in the robot specification. This means the magnitude of the clearing distance is 1-2 meters, which is by the project considered to allow the operator within close proximity of the robot during operation.

The second area is *Functionality*. This area covers the necessary abilities of the motion capture and robot solution to successfully record and recreate the movement of a human kick. The verification testing showed that all criteria related to the robot solution was achieved. The robot was able to repeatedly activate the FMDM sensor by running automated test programs. The robot movement triggering the sensor was based on an imported trajectory and could be performed at different speeds. The

appearance of the robot was subjectively similar to the lower leg of a human with interchangeable shoe attachments and a jeans leg. Due to limitations in RobotStudio, the original speed and orientation of the shoe along the recorded foot trajectory could not be included in the data import generating the robot movement. However, these parameters of the trajectory were manually configured in RobotStudio before downloading the robot program to the physical robot. By utilizing the VR environment in Unity and creating scrips to implement the motion capture recordings, a data set enabling an identical reproduction of the trajectory was generated. The process of recording human motion did not impose any restriction on the user while capturing movement in three dimensions.

The third area is *Reliability*. By recording the full kick trajectory of the forward and backward motion and using it as input for the robot movement, it was never a risk to block the opening or closing of the tailgate with the robot. This is because the robot moves from the potential area of collision with the tailgate after triggering the FMDM sensor. For the motion recording, any body type can attach the HTC Vive Tracker to the foot and use it for recording a kick. The logic of the data recording is based on the relative movement of the GameObject from its initial position, as described in Section 3.4.1. Due to this, any positional changes in the VR software or used equipment will not affect the resulting trajectory. This makes the motion capture process and its output repeatable for different users and test environments.

The fourth area, *Durability*, targets the robot solution and how the design enables consistent output despite changing test vehicles and test locations. First, when executing the same robot programs the robot will perform identical movements. Second, by placing the robot an on an EU pallet, it allows for accurate positioning in relation to a variation of test vehicles by manual measuring and positioning of the pallet. The fifth area, *Ergonomics*, addresses that the robot solution should be able to easily integrate to the existing work place at Volvo Cars. The current robot of model IRB1200 does not impose a disturbance to the surrounding workplace.

5.2 Design of final prototype

When designing the the final setup for the human leg replication, it became apparent that using a solid leg representation was not suitable. As described in Section 3.6.2.3, one reason for this is the changing angle between the lower leg and the foot during a kick. Without introducing a joint, it is not possible to achieve a varying angle when using solid materials. When deciding which position the setup should be the primarily designed for, it is the extended position underneath the car's bumper that is the most important. This is when the leg replication comes in contact with the motion detection of the FMDM sensor. The robot arm is in an extended position, meaning a wide angle between the feet and the lower leg, in this situation. When the robot arm is stationed in its home position before a kick or during the initial and final part of the trajectory, the appearance of the leg replication serves no function as it is not affecting the FMDM sensor. Adaptation to other aspects of the solution became the priority for the leg replication at these sequences of the

robot movement. As can be seen in Figure 3.28a *Robot in home position* during the verification testing, the angle between the robot arm and the shoe is less than 90 degrees. In order for the trajectory from the human motion capture to be reproduced by the robot without manually reducing the length of the trajectory, the robot had to use most of its reach to execute the movement. A home position as close as possible to the robot center and the EU pallet was employed during the testing. The final setup for the human leg replication was allowed to be compressed in its home position. Due to the flexible material of the cylinder placed inside the jeans bag and the support the shoe provided to hold the cylinder in place, the shape of the lower leg remained intact when returning to an extended position. This is shown in Figure 3.28b *Robot in extended position* during the verification testing.

The design using a bracket for attaching a shoe to the robot constitutes a flexible solution. When testing is performed with the robot, any open shoe with a flat sole can be used in the test setup. When combining the flexible human leg replication with the ability record kicks performed by different users, the robot forms a credible solution to replicate the movement of various persons activating the FMDM sensor.

5.3 Involving ABB Robotics

The developed solution uses both hardware and software applications provided by ABB Robotics. In order for Volvo Cars to continue using the final prototype, they are dependent to maintain the services and the robot that have been available during the execution of the project. The probability of this happening is high. Volvo Cars and ABB Robotics did already have an existing collaboration at the start of the project. A range of different ABB robot models are used in Volvo Cars' manufacturing plants all over the world. Involving the company during the project was therefore not considered to constitute any major risk. In retrospect, involving ABB Robotics promoted the product development process in several areas. First and foremost, the company provided the project with a requested robot within a short time period from when asked for. This enabled an extensive amount of time to develop different aspects of the prototype. One example of these included exploration of the different parameters, such as speed and starting position of the trajectory, used for programming of the robot movements. A second was the adaptation of the robot appearance in terms of geometry and materials to enable satisfactory sensor activation of the FMDM. Moreover, ABB Robotics provided the project with work space, storage room for the robot and free access to tools and equipment at their facility. This eliminated any issues related to accessibility of necessities and all focus could be spent on problem solving and finding creative solutions for the prototype. A third area where involving ABB Robotics was advantageous was related to the work with RobotStudio. As RobotStudio is ABB's own software, it was easier to get in contact with software developers working with the program. Involving experts was a necessary step of getting access to and refine the RobotStudio add-in allowing import of coordinates based on a CSV file, which is used in the final prototype.

6
Conclusion

The desired functionality of the final prototype was to enable repetitive, identical kicks that mimic those recorded from a human. This is due to the current absence of methods to perform objective testing and calibration of the FMDM sensor at Volvo Cars. In order for the sensor to register the prototype as an approaching human lower leg during a kick, it needs to match the properties speed, material and geometry to those similar of a human.

The final prototype incorporates a motion tracking system and a robot to perform the kicks. The optical, infrared motion capture system is based on the VR platform Unity to enable the movement of a human foot during a kick to be recorded. The motion capture system is not limited to specific motions, meaning that all foot movement during a recording is captured. The captured Cartesian coordinates are post processed and later imported to the offline robot programming software RobotStudio to generate the robot movements. The robot movement consists of an identical trajectory to that previously performed by a human. The appearance of the robot is adapted to mimic a human lower leg by adding a shoe and jeans material to the setup.

The robot movement is verified to have the desired function through quantitative testing. The testing showed that 300 identical kicks, in terms of speed and trajectory, did not generate 100% motion trigger activations of the FMDM sensor and thus opening or closing the tailgate, but instead 83%. The result forms the base of future development work where Volvo Cars will strive to achieve 100% sensor activation by repeated, identical kicks of a robot movement that should generate a motion trigger of the sensor.

The process of reproducing human movement with a robot is achieved through three steps. These include human motion capture, processing of raw data and generation of a robot program. The process is designed to minimize the amount of manual work required in each step. This allows engineers at Volvo Cars with limited previous knowledge of programs used for motion tracking and robot programming to perform the process of replicating human movement with a robot.

7
Recommendations

There are a number of areas that should be addressed by Volvo Cars with continued use of the proposed final solution in the project. The solution is considered a prototype, created from a product development process and constitutes a starting point for further development.

The area that should be an initial focus is how to develop the use cases for the solution. The project objectives did not include creating a complete test or calibration program where the prototype is used. It is thus required to define and come up with test programs that takes advantage of the functionality of the prototype in an appropriate way. An suggested starting point of future work will be to perform a large number of recordings and thus create a library of kicks available for later use. This library should contain an appropriate range of variations related to kick trajectory, gender, height and shoe size of the user, initial distance from car's bumper, vehicle model and any other property that adds to a diverse data set. Once a kick is recorded and later imported to RobotStudio, the user can easily create and modify test programs by changing which kicks to include, the number of repetitions and the speed of each kick. This will act as a tool to develop automated test programs containing objectively identical kicks. These test programs should be used during future calibration, reliability and benchmark testing.

A second area to investigate is whether the current solution of the lower leg has a representative geometry compared to a real human leg and shoe. This is relevant knowledge when performing different test programs with the prototype. The current solution is validated to consistently activate the FMDM sensor during operation, but the signal processing from the sensor itself is not known. This information is managed by the supplier of the sensor unit. Volvo Cars should work with the supplier to validate that the sensor output from the prototype's leg and and a real leg is comparable. There is a risk of unwanted errors in the FMDM sensor calibration if the difference is significant.

The project solution is dependent of products and services provided by ABB Robotics. A recommendation is to continue the already established communication of project related topics between Volvo Cars and ABB Robotics. This will facilitate the process of solving potential future issues related to the software or hardware for the prototype. If Volvo Cars wants to expand the use cases and introduce multiple or different sized of robots, the established communication will also be of use.

The test sequence of recording human kicks in the project includes a physical car. It would be difficult to replicate a real use case if the kick is performed without any reference of a car's bumper. In order to make the motion recording process more efficient, efforts can be made to eliminate the car. This can be achieved by using only a stand alone bumper and placing it on a stand where correct height above the ground and orientation is secured. This approach will significantly ease the process of using different vehicle model geometries in the recording process, as the bumper is the only part that needs to be changed from an already rigged test environment. A second course of action is to make use of VR capabilities. As shown in Figure 3.5, Volvo Cars has released a model of an XC40 Recharge P8 AWD that is available in a VR environment. The model includes functionality to open and close the tailgate by motion triggers. Some development work is required to ensure precise positioning of the user in relation to the car, but the solution allows for a realistic interaction with a car without including a physical version. The VR environment is currently limited to just one vehicle model, but it is likely that Volvo Cars will expand their library with more models in the future.

Bibliography

[1] VolvoCars, *Operating the tailgate with foot movement*, 2020. [Online]. Available: https://www.volvocars.com/uk/support/topics/use-your-car/car-functions/operating-the-tailgate-with-foot-movement (**urlseen** 28/04/2021).

[2] J. Power, *U.s. initial quality study (iqs)*, 2020. [Online]. Available: https://www.jdpower.com/business/automotive/us-initial-quality-study-iqs (**urlseen** 28/04/2021).

[3] C. Ott, D. Lee **and** Y. Nakamura, "Motion capture based human motion recognition and imitation by direct marker control", **january** 2009, **pages** 399–405. DOI: 10.1109/ICHR.2008.4755984.

[4] H.-H. Jung, M.-K. Kim **and** J. Lyou, "Realization of a hybrid human motion capture system", **in** *2017 17th International Conference on Control, Automation and Systems (ICCAS)*, 2017, **pages** 1581–1585. DOI: 10.23919/ICCAS.2017.8204238.

[5] R. Holubek, D. Delgado Sobrino, P. Košťál **and** R. Roman, "Offline programming of an abb robot using imported cad models in the robotstudio software environment", *Applied Mechanics and Materials*, **jourvol** 693, **pages** 62–67, **december** 2014. DOI: 10.4028/www.scientific.net/AMM.693.62.

[6] C. U. of Technology, *Master's thesis work*, 2021. [Online]. Available: https://student.portal.chalmers.se/en/chalmersstudies/masters-thesis/Pages/masters-thesis-work.aspx (**urlseen** 28/04/2021).

[7] IEEE, *Ieee code of ethics*, 2020. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (**urlseen** 30/04/2021).

[8] VolvoCars, *Volvo cars reports strongest second-half sales in company history on pandemic recovery*, 2021. [Online]. Available: https://www.media.volvocars.com/global/en-gb/media/pressreleases/276308/volvo-cars-reports-strongest-second-half-sales-in-company-history-on-pandemic-recovery (**urlseen** 26/05/2021).

[9] R. W. Mitchell, "A comparative-developmental approach to understanding imitation", **in** *Perspectives in Ethology: Volume 7 Alternatives*, P. P. G. Bateson **and** P. H. Klopfer, Eds. Boston, MA: Springer US, 1987, **pages** 183–215, ISBN: 978-1-4613-1815-6. DOI: 10.1007/978-1-4613-1815-6_7. [Online]. Available: https://doi.org/10.1007/978-1-4613-1815-6_7.

[10] D. Kulić, "Human motion imitation", **in** *Humanoid Robotics: A Reference*, A. Goswami **and** P. Vadakkepat, Eds. Dordrecht: Springer Netherlands, 2019, **pages** 1657–1677, ISBN: 978-94-007-6046-2. DOI: 10.1007/978-94-007-6046-2_34. [Online]. Available: https://doi.org/10.1007/978-94-007-6046-2_34.

[11] MathWorks, *Inverse kinematics (ik) algorithm design with matlab and simulink*, 2021. [Online]. Available: https://www.mathworks.com/discovery/inverse-kinematics.html (**urlseen** 22/05/2021).

[12] K. M. S. Kelly S. Hale, "Handbook of virtual environments", **in** *Design, Implementation, and Applications*, K. M. Stanney, Ed. Mahwah, New Jersey: CRC Press, 2002, **pages** 173–174, ISBN: 978-0-5853-9910-2. [Online]. Available: https://books.google.se/books?hl=sv&lr=&id=sz9ZDwAAQBAJ&oi=fnd&pg=PA163&dq=motion+tracking+definition&ots=2tYoNNcw9T&sig=iDUXunwD3eLrGgWg7-cz2Qv2fu4&redir_esc=y#v=onepage&q=motion%5C%20tracking%5C%20definition&f=false.

[13] R. Solberg **and** A. Jensenius, "Optical or inertial? evaluation of two motion capture systems for studies of dancing to electronic dance music", **january** 2016.

[14] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, M. Elgharib, P. Fua, H.-P. Seidel, H. Rhodin, G. Pons-Moll **and** C. Theobalt, "XNect: Real-time multi-person 3D motion capture with a single RGB camera", 4, **volume** 39, 2020. DOI: 10.1145/3386569.3392410. [Online]. Available: http://gvv.mpi-inf.mpg.de/projects/XNect/.

[15] PhaseSpace, *Impulse x2 motion capture system*, 2021. [Online]. Available: https://www.phasespace.com/impulse-motion-capture.html (**urlseen** 26/05/2021).

[16] M. Brodie, A. Walmsley **and** W. Page, "Fusion motion capture: A prototype system using inertial measurement units and gps for the biomechanical analysis of ski racing", *Sports Technology*, **jourvol** 1, **number** 1, **pages** 17–28, 2008. DOI: https://doi.org/10.1002/jst.6. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jst.6. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jst.6.

[17] R. D. Christ **and** R. L. Wernli, "Chapter 17 - navigational sensors", **in** *The ROV Manual (Second Edition)*, R. D. Christ **and** R. L. Wernli, Eds., Second Edition, Oxford: Butterworth-Heinemann, 2014, **pages** 453–475, ISBN: 978-0-08-098288-5. DOI: https://doi.org/10.1016/B978-0-08-098288-5.00017-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080982885000178.

[18] DigitalMind, *Motion tracking and capture*, 2021. [Online]. Available: https://digitalmind.rs/technology/motion-capture (**urlseen** 26/05/2021).

[19] Antycip, *A brief guide to vr motion tracking technology*, 2021. [Online]. Available: https://steantycip.com/blogs/vr-motion-tracking/ (**urlseen** 23/05/2021).

[20] Xsens-MVN, *The mvn motion capture system hardware range*, 2021. [Online]. Available: https://www.xsens.com/products/mvn-analyze?hsCtaTracking=317368b0-35fa-4dce-b410-de40b523565c%5C%7Ce9ec2617-ba66-47d2-8dc3-c4d5de12a4cd (**urlseen** 05/05/2021).

[21] TESLASUIT, *The suit*, 2021. [Online]. Available: https://teslasuit.io/the-suit/ (**urlseen** 05/05/2021).

[22] VIVE, *Product vive*, 2021. [Online]. Available: https://www.vive.com/eu/product/vive/ (**urlseen** 26/05/2021).

[23] Unity, *Unity platform*, 2021. [Online]. Available: https://unity.com/products/unity-platform (**urlseen** 23/05/2021).

[24] VolvoCars, *3d simulator*, 2021. [Online]. Available: https://developer.volvocars.com/3d/ (**urlseen** 13/05/2021).

[25] Unity, *Gameobject*, 2017. [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html (**urlseen** 23/05/2021).

[26] ——, *Gameobject*, 2021. [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html (**urlseen** 26/05/2021).

[27] Britannica, *Robotics*, 2021. [Online]. Available: https://www.britannica.com/technology/robotics (**urlseen** 09/05/2021).

[28] BostonDynamics, *Products*, 2021. [Online]. Available: https://www.bostondynamics.com/products (**urlseen** 09/05/2021).

[29] ABB, *Robotics*, 2021. [Online]. Available: https://new.abb.com/products/robotics (**urlseen** 09/05/2021).

[30] K. Waldron **and** J. Schmiedeler, "Kinematics", **in** *Springer Handbook of Robotics*, B. Siciliano **and** O. Khatib, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, **pages** 9–33, ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_2. [Online]. Available: https://doi.org/10.1007/978-3-540-30301-5_2.

[31] Robocademy, *Robot kinematics in a nutshell*, 2021. [Online]. Available: https://robocademy.com/2020/04/21/robot-kinematics-in-a-nutshell/ (**urlseen** 09/05/2021).

[32] N. I. of Standards **and** T. (NIST), *4 types of robots every manufacturer should know*, 2017. [Online]. Available: https://www.nist.gov/blogs/manufacturing-innovation-blog/4-types-robots-every-manufacturer-should-know (**urlseen** 23/05/2021).

[33] ABB, *Product specification irb 1300*, 2021. [Online]. Available: https://search.abb.com/library/Download.aspx?DocumentID=3HAC070393-001&LanguageCode=en&DocumentPartId=&Action=Launch (**urlseen** 09/05/2021).

[34] Z. Pan, J. Polden, N. Larkin, S. Van Duin **and** J. Norrish, "Recent progress on programming methods for industrial robots", *Robotics and Computer-Integrated Manufacturing*, **jourvol** 28, **number** 2, **pages** 87–94, 2012, ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2011.08.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584511001001.

[35] ABB, *Robotstudio*, 2021. [Online]. Available: https://new.abb.com/products/robotics/robotstudio (**urlseen** 23/05/2021).

[36] Siemens, *Robotexpert*, 2021. [Online]. Available: https://www.plm.automation.siemens.com/store/en-us/trial/robotexpert.html?gclid=CjwKCAjw-qeFBhAsEiwA2G7Nlyepjyn1EGuwZZbgxNUpE4loIVsRctS2ov1sGpXr9b4lxrYCiSttyRoCSgIQAvD_BwE (**urlseen** 23/05/2021).

[37] Kawasaki, *Simulation olp*, 2021. [Online]. Available: https://robotics.kawasaki.com/en1/products/other/simulation-OLP/ (**urlseen** 23/05/2021).

[38] ABB, *Cobots*, 2021. [Online]. Available: https://cobots.robotics.abb.com/en/ (**urlseen** 23/05/2021).

[39] ValveSoftware, *Base stations*, 2021. [Online]. Available: https://www.valvesoftware.com/en/index/base-stations (**urlseen** 21/05/2021).

[40] VIVE, *Vive tracker*, 2021. [Online]. Available: https://www.vive.com/us/accessory/vive-tracker/ (**urlseen** 21/05/2021).

[41] Varjo, *Varjo announces vr-1, world's first human eye-resolution vr headset for industrial use*, 2021. [Online]. Available: https://varjo.com/press-release/varjo-announces-vr-1-worlds-first-human-eye-resolution-vr-headset-for-industrial-use/ (**urlseen** 21/05/2021).

[42] Teslasuit, *Academic*, 2021. [Online]. Available: https://teslasuit.io/academic/ (**urlseen** 26/05/2021).

[43] V. Les, *Volvo v60 - 2018*, 2018. [Online]. Available: https://grabcad.com/library/volvo-v60-2018-1 (**urlseen** 21/05/2021).

[44] Robario, *Wheel*, 2021. [Online]. Available: https://grabcad.com/library/wheel-709 (**urlseen** 21/05/2021).

[45] M. Asan, *Adidas xtremetrek*. 2020. [Online]. Available: https://grabcad.com/library/adidas-xtremetrek-1 (**urlseen** 21/05/2021).

A

Robot specification

Serial number:	1200-104560
Current state:	Customer Production
Factory warranty status:	Warranty

Order date:	2019-10-21
Created in WebConfig:	2019-11-15
Delivered from factory:	2019-11-20
Factory warranty expires:	2021-06-03
LABB warranty expires:	n/a

MODULE CONFIGURATION OPTIONS

Option	Description	Value
16-1	Application interface Conn. to	Cabinet
57-1	Cabinet colour	ABB standard
94-1	Conn. of Parallel Comm.	7m
200-1	Kortplatser	Qty = 1
209-202	Manipulator colour	ABB Graphite White std
210-2	Manipulator cable - Length	7m
283-41	Product	IRB 1200
287-4	Protection	Standard
334-1	Signs on manipulator	ABB
431-1	Connector kits Upper arm	Yes
435-122	Variant	IRB 1200-5/0.9
438-1	Warranty	Standard Warranty
447-10	Yearmodel	M2004
448-2	Delivery project	SE Standard
545-1	System test	Yes
608-1	World Zones	Yes
632-1	Service Info System	Yes
685-2	Robotware Version	RW 6
698-1	Cabinet parts	Yes
699-1	Manipulator	Yes
700-8	Controller variants	Compact 2nd generation
701-1	FlexPendant	FlexPendant 10 m
709-1	DeviceNet™ m/s	Single ch
716-1	Digital 24V 16In/16Out	Qty = 1
751-1	Drive system	58A 262V
772-2	External axes Yes/No	External axes No
800-2	Release variant	Design 2006
803-1	Media & Communication	Parallel & Air
888-2	PROFINET IO	PROFINET IO m/s SW
996-1	Safety Module	Safety Module
997-2	Safety Network	PROFIsafe F-Host&Device

Figure A.1: Specifications for the robot IRB1200

B

C# Scripts

B.1 Motion tracking Teslasuit

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Events;
using System;
using System.IO;

public class Motion_Tracking_Suit : MonoBehaviour
{
    string timeStamp = DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss");
    public StreamWriter srPos;
    double x, y, z;
    public int count = 0;
    public double initPosx;
    public double initPosy;
    public double initPosz;
    public bool writing;

    // Start is called before the first frame update
    void Start()
    {
        string pathPos = @"C:\Users\SimRig\wkspaces\TestTesla\LegData\LegPosition" + timeStamp +
".txt";
        x = 0;
        y = 0;
        z = 0;

        initPosx = transform.position.x;
        initPosy = transform.position.y;
        initPosz = transform.position.z;
        srPos = File.CreateText(pathPos);
    }
```

```
    // Update is called once per frame
    void Update()
    {

        writing = GameObject.Find("Record").GetComponent<ClickRecord>().writing;

        x = initPosx - transform.position.x;
        y = initPosy - transform.position.y;
        z = initPosz - transform.position.z;

        string xs = string.Format("{0:0.0000}", x);
        string ys = string.Format("{0:0.0000}", y);
        string zs = string.Format("{0:0.0000}", z);

        if (writing)
        {
            srPos.WriteLine(xs.ToString() + "," + ys.ToString() + "," + zs.ToString() + ",");
        }
    }
    void OnApplicationQuit()
    {
        srPos.Close();
    }
}
```

B.2 Motion tracking HTC Vive Tracker

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Events;
using System;
using System.IO;

public class ClickRecord : MonoBehaviour
{
    string timeStamp = DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss");
    string pathPos;
    float x, y, z;
    public float initPosx;
    public float initPosy;
    public float initPosz;
    public float xinit;
    public float yinit;
    public float zinit;
    public StreamWriter srPos;
    public StreamWriter srRot;
    public Button First;
    public Button Second;
    public Button Third;
    public bool eventTimer;
    public Text text;
    public bool writing;
    public bool initValue;
    public GameObject gameObj;
    public float total_time;
```

```
// Start is called before the first frame update
void Start()
{
    pathPos = @"C:\Users\SimRig\wkspaces\TestTesla\TrackerData\TrackerPosition" + timeStamp +
".txt";
    Third.onClick.AddListener(TaskOnClick);
    First.onClick.AddListener(TaskOnClick);
}

// Update is called once per frame
void Update()
{
    x = xinit + initPosx - gameObj.transform.position.x;
    y = yinit + initPosy - gameObj.transform.position.y;
    z = zinit + initPosz - gameObj.transform.position.z;

    string xs = string.Format("{0:0.0000}", x);
    string ys = string.Format("{0:0.0000}", y);
    string zs = string.Format("{0:0.0000}", z);

    if (writing)
    {
        if (initValue)
        {
            srPos.WriteLine(xinit.ToString() + "," + yinit.ToString() + "," + zinit.ToString() + ",");
            initValue = false;
        }
        else
        {
            total_time += Time.deltaTime;
            srPos.WriteLine(xs.ToString() + "," + ys.ToString() + "," + zs.ToString() + ",");
```

```
        }
      }
    }

    void TaskOnClick()
    {

      writing = true;
      srPos = File.CreateText(pathPos);
      text.text = "You are now recording !";

      initPosx = gameObj.transform.position.x;
      initPosy = gameObj.transform.position.y;
      initPosz = gameObj.transform.position.z;
      xinit = 0;
      yinit = 0;
      zinit = 0;
      initValue = true;
    }

    void OnApplicationQuit()
    {
      //srPos.Close();
      string time = string.Format("{0:0.00}", total_time);
      if (writing) {
      Debug.Log("You kicked for " + time + " seconds");
      }
    }
}
```

C

Python Scripts

C.1 Notebook in Google Colaboratory during initial testing

```
from mpl_toolkits import mplot3d
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import csv
from mpl_toolkits.mplot3d.axes3d import Axes3D
from mpl_toolkits.mplot3d.proj3d import proj_transform
from matplotlib import cm
from mpl_toolkits.mplot3d import proj3d
from matplotlib.patches import FancyArrowPatch


class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M
)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)

def cleanData(data):
  data2=[]
  for i in range(len(data)-1):
      if (data[i]!=data[i+1]):
          data2.append(data[i])
  return data2

def UnityToPython(fileName):
  results = []
  with open(fileName) as csvfile:
      reader = csv.reader(csvfile, quoting=csv.QUOTE_NONNUMERIC) # chan
ge contents to floats
      for row in reader: # each row is a list
          results.append(row)
  return results


def plotBothTrajectories(leg,track):
  fig = plt.figure(figsize=(20, 15))
  ax = fig.add_subplot(111, projection='3d')
  for i in range(len(leg)):
    if (i!=len(leg)-1):
      ax.scatter(leg[i][0], leg[i][1], leg[i][2], s=80, c='red', marker
='o')
```

```
      arw = Arrow3D([leg[i][0],leg[i+1][0]],[leg[i][1],leg[i+1][1]],[le
g[i][2],leg[i+1][2]], arrowstyle="->", color="blue", lw = 2, mutation_s
cale=25)
      ax.add_artist(arw)
  for i in range(len(track)):
    if (i!=len(track)-1):
      ax.scatter(track[i][0], track[i][1], track[i][2], s=80, c='yellow
', marker='*')
      arw2 = Arrow3D([track[i][0],track[i+1][0]],[track[i][1],track[i+1
][1]],[track[i][2],track[i+1][2]], arrowstyle="->", color="black", lw =
 2, mutation_scale=25)
      ax.add_artist(arw2)
  ax.set_title('Lower leg and Tracker movements in 3D over the time',fo
ntsize=25,)
  ax.set_xlabel('X', fontsize=10, labelpad=20)
  ax.set_ylabel('Y', fontsize=10, labelpad=20)
  ax.set_zlabel('Z', fontsize=10, labelpad=20);

def PositiveTwoData(array1,array2):
  data1=[]
  data2=[]
  x1=[sub_list[0] for sub_list in array1]
  y1=[sub_list[1] for sub_list in array1]
  z1=[sub_list[2] for sub_list in array1]

  x2=[sub_list[0] for sub_list in array2]
  y2=[sub_list[1] for sub_list in array2]
  z2=[sub_list[2] for sub_list in array2]

  minx=min(min(x1),min(x2))
  miny=min(min(y1),min(y2))
  minz=min(min(z1),min(z2))

  if(minx<0):
    x1=x1+abs(minx)*np.ones_like(x1)
    x2=x2+abs(minx)*np.ones_like(x2)
  if(miny<0):
    y1=y1+abs(miny)*np.ones_like(y1)
    y2=y2+abs(miny)*np.ones_like(y2)
  if(minz<0):
    z1=z1+abs(minz)*np.ones_like(z1)
    z2=z2+abs(minz)*np.ones_like(z2)

  for i in range(len(x1)):
    dat1=[x1[i],z1[i],y1[i]]
    data1.append(dat1)
  for j in range(len(x2)):
    dat2=[x2[j],z2[j],y2[j]]
```

```
    data2.append(dat2)

  return data1, data2


leg=UnityToPython("SweepLeg2.txt")
traj=cleanData(leg)
track=UnityToPython("SweepTrack2.txt")
traj2=cleanData(track)
traj,traj2=PositiveTwoData(traj,traj2)
plotBothTrajectories(traj,traj2)
```

C.2 Notebook in Google Colaboratory during detailed prototype design and construction

```
from mpl_toolkits import mplot3d
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import csv
from mpl_toolkits.mplot3d.axes3d import Axes3D
from mpl_toolkits.mplot3d.proj3d import proj_transform
from matplotlib import cm
from mpl_toolkits.mplot3d import proj3d
from matplotlib.patches import FancyArrowPatch


class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M
)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)


def cleanData(data):
  data2=[]
  for i in range(len(data)-1):
      if (data[i]!=data[i+1]):
          data2.append(data[i])
  return data2


def plotTrajectory(traj,bool=True):
  fig = plt.figure(figsize=(20, 15))
  ax = fig.add_subplot(111, projection='3d')
  for i in range(len(traj)):
    if (i!=len(traj)-1):
      ax.scatter(traj[i][0], traj[i][2], traj[i][1], s=80, c='red', mar
ker='o')
      arw = Arrow3D([traj[i][0],traj[i+1][0]],[traj[i][2],traj[i+1][2]]
,[traj[i][1],traj[i+1][1]], arrowstyle="->", color="blue", lw = 2, muta
tion_scale=25)
      ax.add_artist(arw)
  if(bool==True):
    ax.set_title('Lower leg movements in 3D over the time',fontsize=25,
)
  else:
```

```
    ax.set_title('Tracker movements in 3D under the tailgate',fontsize=
25,)
  ax.set_xlabel('X', fontsize=10, labelpad=20)
  ax.set_ylabel('Z', fontsize=10, labelpad=20)
  ax.set_zlabel('Y', fontsize=10, labelpad=20);


def UnityToPython(fileName):
  results = []
  with open(fileName) as csvfile:
      reader = csv.reader(csvfile, quoting=csv.QUOTE_NONNUMERIC) # chan
ge contents to floats
      for row in reader: # each row is a list
          results.append(row)
  return results


def Switch(traj):
  data_norm=[]
  x=[sub_list[0] for sub_list in traj]
  y=[sub_list[1] for sub_list in traj]
  z=[sub_list[2] for sub_list in traj]
  for i in range(len(y)):
    if (y[i]<=0):
      y[i]=abs(y[i])
    else:
      y[i]=-y[i]
  for i in range(len(x)):
    dat=[x[i],y[i],z[i]]
    data_norm.append(dat)
  return data_norm


def ValuesToMM(traj):
  traj=Switch(traj)
  trajl=[]
  for i in range(len(traj)):
    trajl.append([traj[i][0]*1000,traj[i][1]*1000,traj[i][2]*1000]) #XY
Z
  return(trajl)


def CleanDataToCSV(traj,filename):
  filename=filename[:-4]
  with open(filename + ".csv", 'w') as f:
    for i in range(len(traj)):
      f.write(str(traj[i][0])+","+str(traj[i][2])+","+str(traj[i][1])+"
\n"  ) #XZY
```

```
def UnityToRobotStudio(filename):
  leg=UnityToPython(filename)
  traj=cleanData(leg)
  traj=Switch(traj)
  traj=ValuesToMM(traj)
  plotTrajectory(traj,bool=False)
  CleanDataToCSV(traj,filename)


UnityToRobotStudio("SweepHanna1.txt")
```

D

RobotStudio programs

D.1 Generated code from manually created robtargets

```
MODULE Module1
  CONST robtarget Target_1:=[[1064.93497425,-
1.581796012,214.200960295],[0.011884697,-0.398162114,0.02186256,-0.916977488],[-
1,0,2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  CONST robtarget
Target_2:=[[1300.851005599,68.229488322,214.200931536],[0.011884726,-
0.398161804,0.021862617,-
0.916977621],[0,0,2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  CONST robtarget
Target_3:=[[1360.57961668,171.08921154,285.639769741],[0.073106901,-0.361683411,-
0.057805188,-0.927630881],[0,-1,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  CONST robtarget
Target_4:=[[1404.636042239,261.07601401,283.480502656],[0.076108519,-0.310694132,-
0.053722878,-0.945933667],[0,-1,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  CONST robtarget
Target_5:=[[1286.311056447,358.248878185,311.333981979],[0.078753192,-0.408287155,-
0.067284827,-0.906957709],[0,0,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  CONST robtarget
Target_6:=[[1039.051086109,358.248904382,373.190699876],[0.078753233,-0.40828716,-
0.06728484,-0.906957702],[0,0,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

  PROC LegSweep()
    MoveL Target_1,v800,z100,MyTool\WObj:=wobj0;
    MoveL Target_2,v600,z100,MyTool\WObj:=wobj0;
    MoveL Target_3,v500,z100,MyTool\WObj:=wobj0;
    MoveL Target_4,v400,z100,MyTool\WObj:=wobj0;
    MoveL Target_5,v500,z100,MyTool\WObj:=wobj0;
    MoveL Target_6,v600,z100,MyTool\WObj:=wobj0;
    MoveL Target_1,v800,z100,MyTool\WObj:=wobj0;
  ENDPROC
ENDMODULE
```

D.2 Generated code from add-in used for verification testing

```
MODULE Module1
    PERS speeddata mySpeed:=[800,500,1000,5000];
    TASK PERS wobjdata Straight:=[FALSE,TRUE,"",[[282,-100,-
40],[0.707106781,0,0.707106781,0]],[[0,0,0],[1,0,0,0]]];
    CONST robtarget
TargetST_1:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_2:=[[-0.9,0.3,0.5],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_3:=[[-1.6,-0.1,1.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_4:=[[-2,-0.2,2.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_5:=[[-2.7,-0.3,4],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_6:=[[-3.5,-0.3,5.5],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_7:=[[-4.9,-0.1,7.4],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_8:=[[-6.4,0,9.3],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_9:=[[-8.2,0.5,12.1],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_10:=[[-10,1,14.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_11:=[[-12.2,2.5,17.6],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_12:=[[-14,4.4,21.2],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_13:=[[-15.9,7.3,24.6],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_14:=[[-18.6,10.7,27.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_15:=[[-20.7,13.8,33.4],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_16:=[[-22.2,15.9,40.4],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_17:=[[-24.3,18,50.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_18:=[[-26.5,21.2,60.9],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_19:=[[-28.9,23.7,73.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_20:=[[-31.5,25.4,87.3],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_21:=[[-34.2,26.8,104.3],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_22:=[[-37.1,28.9,121],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_23:=[[-40.5,30.4,140],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    CONST robtarget TargetST_24:=[[-44.1,32,159.3],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_25:=[[-46.9,34.4,180.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_26:=[[-49,37.5,201.9],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_27:=[[-51.1,40.8,225],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_28:=[[-54,44.9,247.8],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_29:=[[-58.4,50,271.1],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_30:=[[-62.8,55.6,294.7],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_31:=[[-67.8,61.6,317.2],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_32:=[[-73.4,68.1,340.2],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_33:=[[-79.2,74.8,362.8],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_34:=[[-84.6,81.7,384.6],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_35:=[[-89.7,89.3,406.2],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_36:=[[-94.5,97.1,426.6],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_37:=[[-
99.5,105.6,446.1],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_38:=[[-
103.9,114.7,464.3],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_39:=[[-
106.7,125.6,480.3],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_40:=[[-
109.3,136.8,496.3],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_41:=[[-
110.4,148.5,508],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_42:=[[-
109.7,160.8,520],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_43:=[[-
108.7,170.9,528.9],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_44:=[[-
108.3,180.1,538.5],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_45:=[[-
107.8,187.1,547.3],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_46:=[[-
107.2,191.5,555.5],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_47:=[[-
107.1,195.9,563.6],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_48:=[[-
107.1,200.2,570.2],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    CONST robtarget TargetST_49:=[[-
106.7,204.8,575.5],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_50:=[[-
106.5,209.5,579.8],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_51:=[[-
106.6,213.2,583.4],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_52:=[[-
107,216.1,587.4],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_53:=[[-
107.8,218.4,591.3],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_54:=[[-
108.6,220.3,595.1],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_55:=[[-
109.2,222.4,598.9],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_56:=[[-
109.7,224.9,602.5],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_57:=[[-
109.7,227.9,605.6],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_58:=[[-
84.9,134.3,458.7],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_59:=[[-
80.5,122.9,440.6],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_60:=[[-
75.2,111.9,421.2],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_61:=[[-
71.1,101.4,401.7],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_62:=[[-65.3,91.7,382.5],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_63:=[[-59.7,82.9,362.3],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_64:=[[-54.3,74.1,343],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_65:=[[-49.4,65.9,323.9],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_66:=[[-43.8,58.9,303.8],[1,0,0,0],[-
1,1,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_67:=[[-38,52.8,284.1],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_68:=[[-32.8,46.7,264.4],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_69:=[[-28.7,43.1,247.1],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_70:=[[-24.2,39.6,227.6],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_71:=[[-20.3,35.8,209.5],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_72:=[[-16.3,31.7,191.2],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_73:=[[-12.7,27.9,171.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    CONST robtarget TargetST_74:=[[-9.9,25.8,154.3],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_75:=[[-7.5,23.8,137.5],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_76:=[[-5.6,21.9,121.5],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_77:=[[-3.6,19.9,106.2],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_78:=[[-2.2,18.1,91.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_79:=[[-0.9,16.6,75.4],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_80:=[[-0.4,15.6,61.7],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_81:=[[-0.1,14.4,49.5],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_82:=[[0.4,12.9,38],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_83:=[[0.6,12.1,29.1],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_84:=[[1,11,19.2],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget TargetST_85:=[[1,9,10.9],[1,0,0,0],[-
1,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];


    PROC main()
    mySpeed.v_tcp:=800;
    FOR i FROM 1 TO 100 DO
        TPWrite "Kick: "+valtostr(i);
        Straight_1;
        WaitTime 6;
    ENDFOR
    ENDPROC

    PROC Straight_1()
        MoveJ TargetST_1,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_2,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_3,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_4,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_5,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_6,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_7,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_8,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_9,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_10,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_11,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_12,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_13,mySpeed,z100,tool0\WObj:=Straight;
        MoveL TargetST_14,mySpeed,z100,tool0\WObj:=Straight;
```

```
MoveL TargetST_15,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_16,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_17,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_18,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_19,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_20,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_21,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_22,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_23,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_24,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_25,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_26,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_27,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_28,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_29,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_30,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_31,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_32,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_33,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_34,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_35,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_36,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_37,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_38,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_39,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_40,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_41,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_42,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_43,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_44,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_45,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_46,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_47,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_48,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_49,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_50,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_51,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_52,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_53,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_54,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_55,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_56,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_57,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_58,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_59,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_60,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_61,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_62,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_63,mySpeed,z100,tool0\WObj:=Straight;
MoveL TargetST_64,mySpeed,z100,tool0\WObj:=Straight;
```

```
    MoveL TargetST_65,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_66,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_67,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_68,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_69,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_70,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_71,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_72,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_73,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_74,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_75,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_76,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_77,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_78,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_79,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_80,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_81,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_82,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_83,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_84,mySpeed,z100,tool0\WObj:=Straight;
    MoveL TargetST_85,mySpeed,z100,tool0\WObj:=Straight;
  ENDPROC
ENDMODULE
```

E

Sketch of profile used to create a geometry in CATIA V5

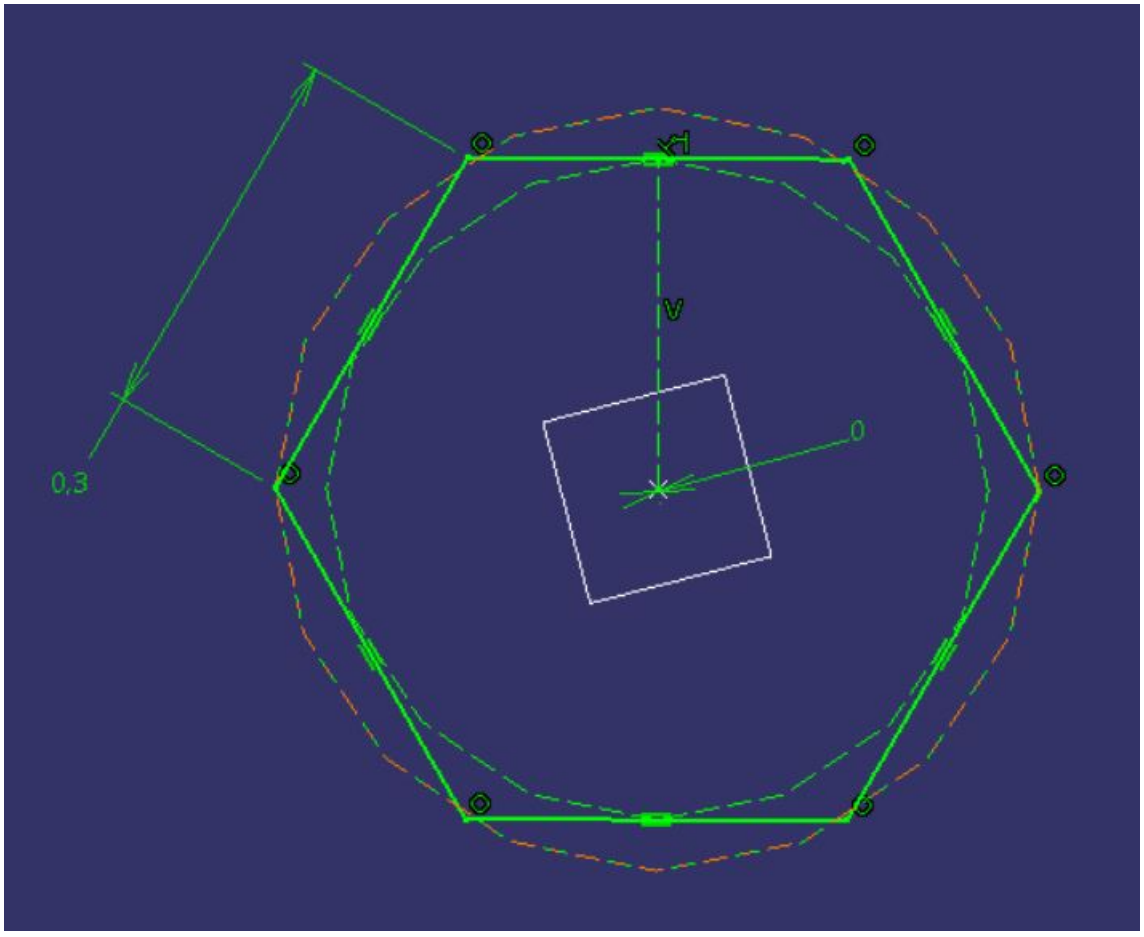


Figure E.1: Sketch of profile used for creating a solid geometry in CATIA V5

F

Data file with coordinates used for verification testing

0.0,2000.0,4000.0
0.0,2000.0,4000.1
0.0,2000.0,4000.0
-0.1,2000.2,4000.0
-0.1,2000.3000000000002,4000.0
0.0,2000.1000000000001,4000.0
0.0,2000.1000000000001,3999.8999999999996
0.1,2000.0,3999.8999999999996
0.0,2000.0,3999.8999999999996
-0.1,2000.0,4000.0
-0.1,1999.9,4000.0
-0.1,1999.9,3999.8999999999996
-0.1,1999.8,3999.8999999999996
-0.1,1999.8,4000.0
-0.2,1999.9,4000.0
-0.2,1999.8,4000.0
-0.2,1999.8,3999.8999999999996
-0.2,1999.8,4000.0
-0.2,1999.7,4000.0
-0.2,1999.7,3999.8999999999996
-0.2,1999.7,4000.0
-0.3,1999.7,4000.0
-0.4,1999.9,4000.0
-0.3,1999.8,4000.0
-0.3,1999.7,4000.0
-0.3,1999.6000000000001,4000.0
-0.3,1999.6000000000001,3999.8999999999996
-0.4,1999.6000000000001,4000.0
-0.4,1999.7,4000.0
-0.4,1999.7,4000.1
-0.4,1999.7,4000.0
-0.3,1999.8,4000.0
-0.4,1999.7,4000.0
-0.4,1999.7,4000.1
-0.5,1999.6000000000001,4000.1
-0.6,1999.7,4000.1
-0.5,1999.6000000000001,4000.1
-0.6,1999.6000000000001,4000.0
-0.6,1999.5,4000.0
-0.7,1999.6000000000001,4000.0
-0.7,1999.5,4000.0
-0.7,1999.6000000000001,4000.0
-0.7,1999.6000000000001,3999.8999999999996
-0.8,1999.6000000000001,3999.8
-0.8,1999.6000000000001,3999.7
-0.9,1999.7,3999.5
-1.1,1999.7,3999.2999999999997

-1.1,1999.9,3999.2000000000003
-1.2,1999.9,3998.8
-1.6,2000.1000000000001,3998.3
-2.0,2000.2,3997.3
-2.7,2000.3000000000002,3996.0
-3.5,2000.3000000000002,3994.5
-4.8999999999999995,2000.1000000000001,3992.6
-6.4,2000.0,3990.7
-8.200000000000001,1999.5,3987.8999999999996
-10.0,1999.0,3985.3
-12.200000000000001,1997.5,3982.4
-14.0,1995.6000000000001,3978.8
-15.9,1992.6999999999998,3975.4
-18.599999999999998,1989.3000000000002,3972.3
-20.7,1986.2,3966.6
-22.2,1984.1,3959.6
-24.299999999999997,1982.0,3949.3
-26.5,1978.8,3939.1
-28.9,1976.3,3926.2999999999997
-31.5,1974.6,3912.7000000000003
-34.2,1973.2,3895.7000000000003
-37.1,1971.1000000000001,3879.0
-40.5,1969.6,3860.0
-44.1,1968.0,3840.7
-46.9,1965.6,3819.3
-49.0,1962.5,3798.1
-51.1,1959.2,3775.0
-54.0,1955.1000000000001,3752.2000000000003
-58.4,1950.0,3728.9
-62.8,1944.3999999999999,3705.2999999999997
-67.8,1938.3999999999999,3682.7999999999997
-73.4,1931.8999999999999,3659.8
-79.2,1925.2,3637.2
-84.6,1918.3,3615.4
-89.7,1910.7,3593.7999999999997
-94.5,1902.9,3573.4
-99.5,1894.4,3553.9
-103.9,1885.3,3535.7
-106.7,1874.4,3519.7
-109.3,1863.2,3503.7
-110.39999999999999,1851.5,3492.0
-109.7,1839.2,3480.0
-108.7,1829.1,3471.1
-108.3,1819.9,3461.5
-107.80000000000001,1812.8999999999999,3452.7000000000003
-107.2,1808.5,3444.5
-107.1,1804.1000000000001,3436.4

-107.1,1799.8,3429.8
-106.7,1795.1999999999998,3424.5
-106.5,1790.5,3420.2
-106.6,1786.8,3416.6
-107.0,1783.9,3412.6
-107.80000000000001,1781.6000000000001,3408.7000000000003
-108.60000000000001,1779.7,3404.9
-109.2,1777.6000000000001,3401.1
-109.7,1775.1,3397.5
-109.7,1772.1,3394.4
-84.9,1865.6999999999998,3541.3
-80.5,1877.1,3559.4
-75.2,1888.1,3578.8
-71.1,1898.6000000000001,3598.3
-65.3,1908.3000000000002,3617.5
-59.7,1917.1000000000001,3637.7000000000003
-54.300000000000004,1925.8999999999999,3657.0
-49.4,1934.1,3676.1
-43.8,1941.1000000000001,3696.2000000000003
-38.0,1947.2,3715.9
-32.800000000000004,1953.3,3735.6
-28.7,1956.9,3752.9
-24.2,1960.3999999999999,3772.4
-20.299999999999997,1964.2,3790.5
-16.299999999999997,1968.3,3808.8
-12.7,1972.1,3828.3
-9.9,1974.2,3845.7
-7.5,1976.2,3862.5
-5.6,1978.1,3878.5
-3.6,1980.1,3893.8
-2.2,1981.9,3908.3
-0.9,1983.4,3924.6
-0.4,1984.3999999999999,3938.2999999999997
-0.1,1985.6000000000001,3950.5
0.4,1987.1000000000001,3962.0
0.6,1987.9,3970.9
1.0,1989.0,3980.7999999999997
1.0,1991.0,3989.1
1.2,1993.6000000000001,3996.5
1.5,1997.0,4002.3999999999996
2.2,2001.0,4008.1
2.9,1997.2,4012.4000000000005
3.4,1999.6000000000001,4015.2999999999997
4.5,1999.5,4014.9999999999995
3.8,1999.3000000000002,4014.5
4.8999999999999995,1999.4,4014.7000000000003
5.8,1999.8,4014.7000000000003

5.4,1999.1000000000001,4014.9999999999995
5.5,1998.5,4016.1
5.3,1998.6,4015.7
5.3,1998.3999999999999,4016.1
5.2,1998.5,4015.7999999999997
5.0,1998.6999999999998,4016.0
4.6,1998.6999999999998,4016.0
4.6,1998.6999999999998,4016.2000000000003
4.6,1998.6,4015.9
4.5,1998.8,4016.1
4.6,1998.8999999999999,4015.7999999999997
4.5,1998.8999999999999,4015.9
4.4,1999.0,4015.7999999999997

DEPARTMENT OF INDUSTRIAL AND MATERIALS SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY