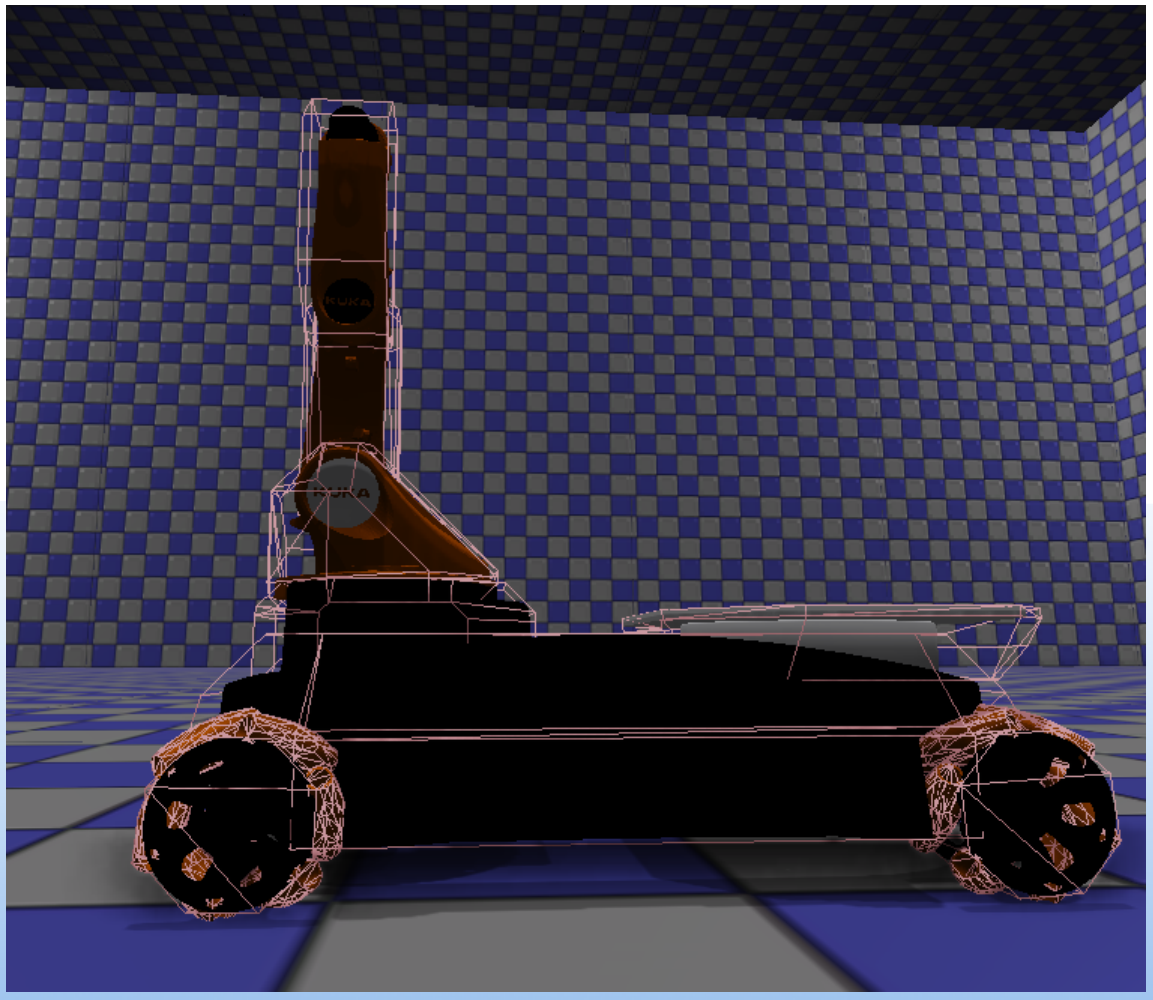

A KUKA youBot simulation in USARSim

Freddy de Greef

Bachelor Informatica

Universiteit van Amsterdam

KUKA youBot simulation

Freddy de Greef

June 28, 2015

Supervisor: Arnoud Visser (UvA)

Signed: Arnoud Visser (UvA)

Abstract

This thesis presents a model to describe the movements of an omnidirectional vehicle platform based on Mecanum wheels. This model will be validated by a simulation of the KUKA youBot in USARSim, an open source high fidelity robot simulator that can be used for research and education. The KUKA youBot offers an experimental platform which bridges the gap between industrial robotics of today and service robotics of tomorrow. The robot possesses four Mecanum wheels providing the robot an omnidirectional mobile platform. The omnidirectional aspect will give the KUKA youBot high maneuverability by allowing the vehicle to move in every direction of the plane. This thesis describes the USARSim architecture, the kinematics of the Mecanum wheels and validates these kinematics in the USARSim environment by comparing the behavior of the simulated KUKA youBot to that of the real one. These kinematics can be extended to other robots with a comparable omnidirectional drive. The results indicate that the movement of the KUKA youBot can be simulated in a realistic way, however, further parameter optimization and experiments testing every possible move are needed. It seems that USARSim could support the behavior of an omnidirectional mobile platform.

Acknowledgements

I would like to express my deepest thanks and gratitude to my advisor, Dr. Arnoud Visser, for his support, enthusiasm for robotics, supervision and advice for this research project.

I would also like to thank my uncle Drs A-Chong Lie and Afra Klarenbeek for their advise for writing this thesis, and Lorenza van den Berg and Henok Tesfai for their help in executing the experiments. Kudos to Mustafa Karaalioğlu for his practical advise for executing the experiments.

Lastly, I would like to acknowledge my parents, Joyce Lie and Fred de Greef, for their continuous support.

Contents

CHAPTER 1

Introduction

This thesis is about a simulator which can be used in robotic research and education. Robotic research provides the ability to keep improving robots, e.g. the aim for a robotic soccer team[1] or a robotic version of the cheetah[2]. The frontier of all known physical robot capabilities is continuously being pushed forward by testing new possibilities. Simulating robotic behavior is a fast and reliable way to find a novelty in the field of robotics. Using a simulator has practical, efficiency and also scientific benefits. A simulation offers a virtual environment which is completely under control. There is no unknown noise coming from the virtual setting. Also, the ability to keep the simulation data stored in logs is supported. Data logs can be kept of sensor information, so absolutely no information is lost during development. Logs are crucial in providing scientific results. They also make debugging and parameter optimization easier. Furthermore running an experiment in a simulation gives the user the ability to share not only the results but also their entire experimental set up. Simulation specifications and classes can be shared, e.g. robot classes, sensor classes or the environment file. Not to mention unlimited access to robotic research, there are no robot malfunctions in the virtual world. However the downside of using a simulation is that a simulation is only an approximation of the reality. High fidelity in the simulation is a necessity for obtaining meaningful scientific results. In order for the simulation results to be valid for implementation on real hardware, the accuracy of the simulation as well as the simulation model of the robot must be validated. Altogether computer simulation of robotic behavior is an essential tool for the development of robotic software.

The simulator used in this thesis is USARSim, short for Unified System for Automation and Robot Simulation. USARSim today is based on the Unreal UDK engine and has the physics of

Figure 1.1: Simulation ran in the USARSim environment [3]

the NVIDIA PhysX engine. USARSim is a validated robotic simulator and the results obtained by this simulation should be representative for the actual robot [15], [9]. USARSim offers a variety of robot classes and sensor classes which form the foundation for any new class to be made. The sensor called groundtruth gives the absolute position of the robot. The simulator also provides a variety of location sensors, each supported with their own simulated noise. For this experiment the groundtruth sensor will be used to eliminate any noise other than the noise in the robotic behavior.

USARSim has experienced wide community acceptance with over 83.000 component downloads to date[4]. USARSim constitutes the simulation engine used to run the Virtual Robots Competition within the RoboCup initiative. RoboCup is an annual scientific event where soccer and rescue competitions take place.

An educational robot found in many robot competitions like the RoboCup@Work competition and the RoCKIn@Work competition is the KUKA youBot. The aim of these competitions is to foster research and development by competing in a wide variety of tasks where robots cooperate with human workers. The KUKA youBot is an educational robot with a high programmability. It is delivered with ROS drivers and is programmable at nearly all levels. The robot consists of a chassis with four Mecanum wheels forming the KUKA youBot base. Two additional robot arms can be placed on top of this base, but it is most commonly used with one arm. The additional arms, the base and the sensors compose the entire robot. The focus of this thesis is only the KUKA youBot base.

Mecanum wheels are used by a wide variety of robots, e.g. the Uranus seen in Figure (1.2), SUMMIT XL HL[5] and the KUKA youBot. These wheels give its platform an omnidirectional aspect. An omnidirectional platform can move in any direction of the plane without having to turn. The usage of the wheels is not limited to robots, they are also found in domestic use or industrial use, e.g. a chair or a forklift. The main concept of the Mecanum wheel is that it has a main wheel with rollers mounted around its periphery. The main wheels of the vehicle are steered by the user, while the rollers are freely movable.

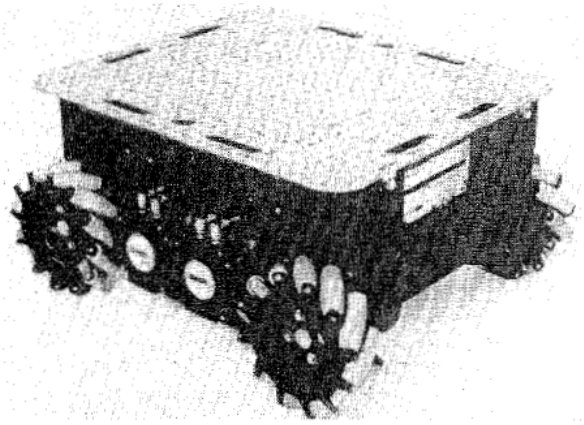

Figure 1.2: Uranus robot uses the Mecanum wheels, 1990 [3]

[1] http://www.robocup2014.org/?page_id=51
[2] http://www.bostondynamics.com/robot_cheetah.html
[3] http://openrdk.SourceForge.net/index.php?n=Tutorials.USARSim
[4] http://SourceForge.net/projects/usarsim/files/stats/timeline?dates=2005-08-04+to+2015-06-02
[5] http://www.robotnik.eu/mobile-robots/summit-xl-hl/

1.1 Research question

The goal for this thesis is to define a model for the movement of an omnidirectional vehicle based on Mecanum wheels. This model will be validated by a simulation of the KUKA youBot base in USARSim. The movement of the real KUKA youBot base will be compared with the movement of the simulated robot.

The implementation will be an approximation of the real robot. This approximation will have every aspect of the wheels modeled as closely to the real one as possible. The results of the movement of both the real and the simulated robot will be compared. This thesis focuses on the following questions:

- Is USARSim able to support the movement of an omnidirectional platform?
  - Does an approximation of the rollers joints reproduce in the movement of the real robot in USARSim?

CHAPTER 2

Related work

Simulation is an important tool for robotic research as well for a great variety of other research areas. In order for the simulation to contribute to the research, it has a few necessities. The simulation is required to provide a realistic representation of the physics that are to be studied. It should also provide a reliable sensor structure and data logging component, in order to obtain simulation data. Results measured in the simulation are crucial in providing scientific results. There are various other robot simulators out there, below a list is provided of robotic simulators that satisfy previously stated requirements.

2.1 Stage

Stage version 3 is the simulator backend to the Player middleware[1]. Stage simulates a two-dimensional world where multiple robots can concurrently act. It is realistic enough for many purposes, finding a balance between fidelity and abstraction. Stage is also very scalable, making the simulator suitable for swarm robotics and related research involving a large robot population [20].

The KUKA youBot is not available in Stage.

2.2 Gazebo

Gazebo is a open source multi-robot simulator for outdoor environments[2]. It can also be the backend of the Player middleware or its own native interface. Like Stage, it is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. Gazebo allows several physics engines, of which the physics engine Open Dynamics Engine is frequently used. Gazebo provides high fidelity and is able to handle complex simulation environments, which allows a wide variety of applications to be possible. The simulator also has an active base of contributers keeping the simulator evolving at a rapid pace [5].

Gazebo provides a KUKA youBot implementation, which uses the kinematic approach. The wheels of the vehicle do not move, only the vehicle moves into the correct direction.

2.3 Webots

Webots is a commercial simulator initially developed to aid software development for the Kephera robots[3]. Nowadays it is widely used for educational purposes. It is a three-dimensional mobile robot simulator and it uses Open Dynamics Engine for physics simulation. Webots is used for research and educational purposes in mobile robotics. The simulator is used in the following research and education areas: mobile robot prototyping, robot locomotion research, multi-agent research, adaptive behavior research, teaching robotics and robot contests [2].The simulator is proprietary however.

Webots provides a detailed KUKA youBot base implementation, which uses the same implementation used in this thesis. The entire robot can be used in this simulator

2.4 SimRobots

SimRobots provides a three-dimensional simulation environment, with a physical model which is based on rigid body dynamics of the Open Dynamics Engine [6]. SimRobots is a robot simulator software package. It was developed in 1990 at the Universität Bremen and the German Research Center for Artificial Intelligence[4]. The simulator was initially designed with the idea of a kinematic, single-user, single-workstation robot simulator. It is being used for further research on autonomous robots. The simulator can also be found among many others within the prestigious group of RoboCup simulators [7].

The youBot is not available in this simulator.

2.5 MORSE

Modular Open Robots Simulation Engine is a three-dimensional open source robotics simulator. MORSE is a python based simulator, which uses the also python based three-dimensional open source modeling program Blender[5]. Simulations are executed on Blenders Game Engine mode using the Bullet physics engine. These components provide the realistic graphics of the simulator [11]. The simulator makes use of a Software In The Loop (SAIL) architecture for exchanging data between MORSE components.

MORSE has no implementation of the KUKA youBot yet [6], but an implementation of the KUKA youBot is currently in progress [7].

[1] http://playerstage.SourceForge.net/?src=stage
[2] http://gazebosim.org/
[3] http://www.cyberbotics.com/
[4] http://www.informatik.uni-bremen.de/simrobot/index_e.htm
[5] https://www.blender.org/
[6] https://github.com/morse-simulator/morse/tree/master/testing/robots
[7] http://www.student.kuleuven.be/~s0212251/

CHAPTER 3

USARSim

USARSim, Unified System for Automation and Robot Simulation, is a high fidelity robot simulator made in 2002. It was initially designed for urban search and rescue simulations but has been developed into a simulator supporting a wide variety of research areas. The simulator is based on the commercial Unreal Tournament game engine. It is open source, therefore the simulator is available for everyone. The open source aspect allows the community to define models for robots or sensors, which can be added to the USARSim models. USARSim can be interfaced with Player, a previously popular middleware used to control many different robots, and with the Mobility Open Architecture Simulation and Tools framework (MOAST), a fully functioning hierarchical control system. USARSim works in a three-dimensional world using the NVIDIA PhysX engine.

It is validated that the USARSim simulation behaves in much the same way as reality [12]. There are also validated models for the movement of several robots in USARSim [9], [16]. Validation of a robot does not mean that the virtual robot moves exactly like the real robot without any error. It means that the movement of the virtual robot is representative for that of the real one. Validation of the simulator and the robot is important in gaining sound scientific results of robot experiments within the virtual environment.

At the creation of USARSim, it only included a few models for differential drive robotic platforms, a restricted set of sensors and the models of three environments. In 2005, following the endorsement of the RoboCup federation USARSim management was taken over by NIST, who moved the project to SourceForge, and reorganized the code base while maintaining most of the original structure. The transition to SourceForge and the involvement of the RoboCup community brought a rapid development in USARSim [15]. RoboCup is an annual international robotics competition with the aim to foster robotic research. The involvement of USARSim in RoboCup will be discussed in the next section.

The simulator was initially designed to support urban search and rescue (USAR) robots and USARSim originally stood for "urban search and rescue simulation". The requirement for a USAR simulator is to simulate an emergency situation with humans being in danger in a complex environment. For example in Figure (3.1), if the emergency is a fire, the fire itself is a threat to humans, but also the smoke is dangerous. Both threats, the humans in need and the robots should behave in a highly realistic way. In order to support these features the simulator requires human-robot interaction (HRI), multi-robot coordination and a high fidelity.

This version of USARSim was based on the Unreal Engine 2. It was created as a modification for the game Unreal Tournament 2004 (UT2004). The UT2004 version supports a wide range of different robots varying from wheeled robots, aerial robots and legged robots. The physics simulation of Unreal Engine 2 is provided by the Karma physics engine. The next version of USARSim moved to Unreal Engine 3 and was created as a modification for the game Unreal Tournament 3. This version of USARSim supported a less wide range of robots and was mainly focused on wheeled robots. This engine uses the NVIDIA PhysX for physics instead of the Karma physics engine. The switch of physics engine caused a rise in processing speed due to improved parallelization, support for more sophisticated models and enhancements to lighting, animation

Figure 3.1: USARSim used for the Rescue Virtual Robot Simulation [1]

and interactions with the simulation environment. The current version of USARSim moved to the Unreal Development Kit (UDK). UDK uses a newer version of Unreal Engine 3 and is monthly updated. One advantage of this version is that the simulator is no longer a modification for a game, which simplifies the requirements for running the simulation. Currently, the latest Unreal engine is the Unreal Engine 4, also created by Epic Games.

3.1 USARSim in robot competitions

USARSim is the basis of the Virtual Robots Competition within the RoboCup initiative. This competition benefits robotic research by focusing tasks on current robotic problems in areas like multi-robot control and mapping. By focusing on these problems the competition provides helpful solutions and insights which benefits the community. The competition has tasks concerning urban search and rescue. The goal of a competing team is to use robots to explore and map the unknown areas of a disaster area and to report as much information as possible to a hypothetical team of first responders in the virtual world. Progress is stimulated by making every participant obligated to release their code to the community. By releasing code to the community, USARSim gains more open source models supporting current robotic problems. Thanks to this, the competition also fosters the development of USARSim [18]. USARSim is designed as a simulation companion to the National Institute of Standards (NIST).

3.2 Game engines in scientific research

The overlap in focus on simulating realistic behavior by both the gaming industry and researchers has created the concept of a scientific simulator build upon a gaming engine. Games have been around for over five decades, starting with 'Space War' made in 1962[2]. In the early days games were written by a small team of programmers or by a creative individual, usually starting from scratch with very few reusable components. The notion that reusable components are useful in programming came in 1968 when Edsger Dijkstra published "Go To Statements Considered Harmful", pleading for a better way of programming [24]. The use of reusable software components is called modular programming. This way of programming has taken great popularity

[1] https://wiki.cc.gatech.edu/robocup/index.php/Rescue_Virtual_Robot_Simulation
[2] http://inventors.about.com/od/sstartinventions/a/Spacewar.htm

and is key in how programmers design their software products nowadays. This also applies for simulated games. Developing a high fidelity simulation costs a lot of resources, meaning game makers can no longer afford making a new three-dimensional from scratch. Gaming companies make several games utilizing reusable components to their maximum potential. The Unreal Engine 3 is the basis for 185 games on several gaming platforms[1]. Game engines are a very modular component. As a result of this separability, game engines can also be purposed for scientific research [21]. Because USARSim uses a continuously improving game engine, "every improvement driven by the ever improving gaming industry translates into USARSim's advantages." [17]

3.3 Architecture

USARSim is built up from three major components: the Unreal Tournament game engine, models and a controller. This is shown in Figure (3.2). The Unreal engine used in the simulator is released by Epic Games. USARSim currently uses the UDK game engine. Communication with the Unreal engine is complicated because of its proprietary communication protocol. However, Gamebots is a modification build by researchers that allows easy communication with the game engine. It opens a socket that allows communication over a TCP/IP network protocol [25].

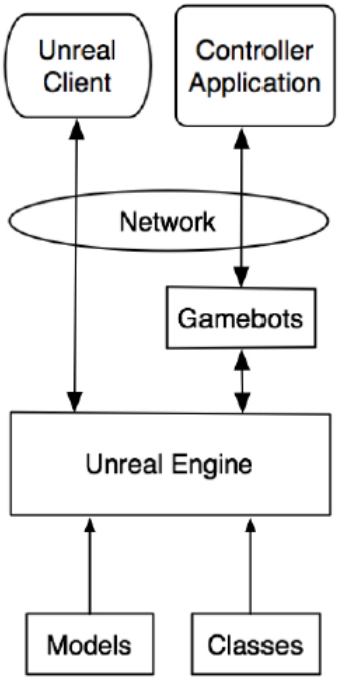

Figure 3.2: USARSim architecture [29]

The models for USARSim are classes, which constitute the building blocks of every robot and sensor. Classes in the Unreal engine are written in UnrealScript and are called the UDK classes. UnrealScript is a custom language which looks a lot like the `C++` language. USARSim classes are written in the same language and provide an addition to the Unreal classes. Together, the UDK classes and the USARSim classes make up the entire class system providing the basis for all models in USARSim.

A robot class specifies what components are used to build the robot. Components are static meshes, joints and robot parameters. Static meshes and joints will be discussed in the next

[1] `http://www.giantbomb.com/unreal-engine-3/3015-86/games/`

section. Robot classes are built upon vehicle classes, which define the way a vehicle is steered, e.g. the vehicle is skid steered.

The sensors classes are made up hierarchically, which will allow the user to add a new sensor by extending from previously defined classes. A subset of all sensor classes is shown in Figure (3.3), which displays the hierarchic division. Sensors have a time-step which specifies the frequency of measurements. One sensor used in the implementation of the experiment is the groundtruth sensor. This sensor measures the value of the robot location and rotation in the x, y, z axes without any noise. Most sensors have noise added to them to match the noise of a real world sensor [30].

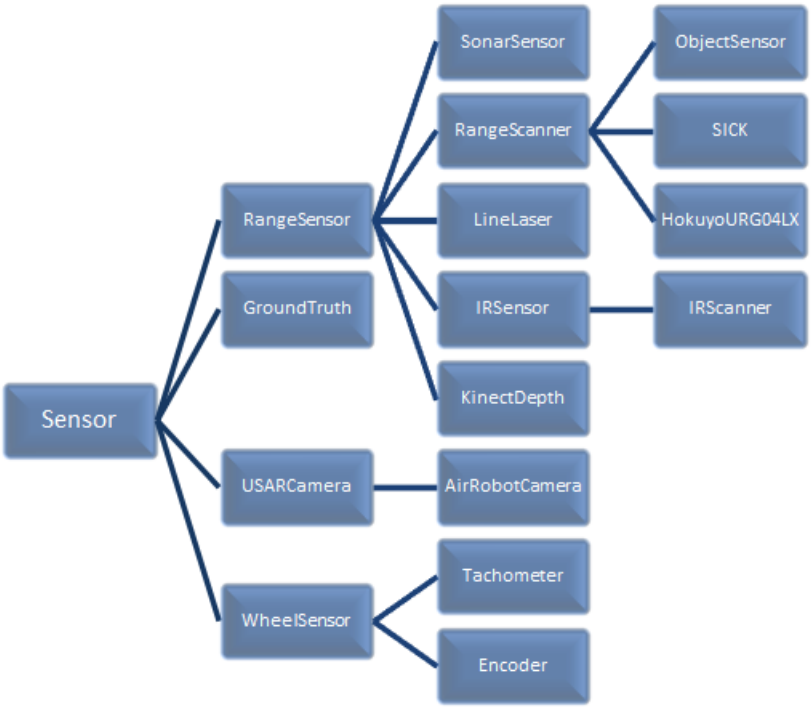

Figure 3.3: Subset of the sensor classes of USARSim

USARSim has a variety of front-ends, which specifies the controller being used. The controller handles the communication of USARSim. Commands for the robot are sent to the game engine. For example it can spawn a robot with an 'INIT' command, and drive this around with 'DRIVE' commands. USARSim sends sensor information via the same channel to the controller, which in turn displays the information to the user. The way the information can be logged depends on the controller choice.

3.4 Simulation

A static mesh is a 3D representation of an object given in vertices, edges and faces. The Unreal engine builds every static mesh through use of triangles.

A rigid body is a static mesh with the physics of the NVIDIA PhysX engine applied to it. In Unreal editor the static meshes are linked to material.

Static meshes can be linked to each other using joints. USARSim classes include joint classes, which can specify the joints being used. A joint in USARSim defines how a rigid body can pivot. A joint is made up by a class. There are a variety of joint classes, e.g. revolutejoint, prismaticjoint and wheeljoint. Joints have a few parameters parent, child, side, location, direction, maximum velocity and stiffness. The joint should always be assigned to a child and a parent. The child is the rigid body that pivots and the parent is the rigid body the child is attached too. Side is important for the steering class. The side parameter puts a label on the joint. For example the skid-steered

class puts wheel speeds on the wheels on the left and the wheels on the right. The controller sends a command DRIVE {Left 1,00}{Right 1,00}. The skid-steered class listens to this drive command and parses the left-speed and the right-speed. The left-speed is assigned to the joints with the side 'Left' and the right-speed is assigned to the joints with the side 'Right'. Direction of a joint has the form $Direction = (X = X\_rotation, Y = Y\_rotation, Z = Z\_rotation)$ where the rotations define the direction of the joint.

Collision specifies the boundary of a rigid body. Collision takes form in a static mesh attached to a rigid body. As seen in Figure (3.4) the collision is represented as a static mesh.

Physical materials are used to define the response of a physical object when interacting dynamically with the world. In the Unreal engine this is defined as a collection of physical variables. Some noteworthy variables are AngularDampening, Friction, LinearDampening and Restitution. The value for AngularDampening is the amount of resistance a physical object will have while rotating. The Friction value defines how much resistance to movement the object will have when touching or sliding along another surface. It commonly has a range from 0 to 1. If the value equals zero, the material is frictionless and if the value is 1, the material has a rough surface. The Linea Dampening value is the amount of resistance an object will have when moving in the world. This value is not the same as friction. The Restitution value is the amount of 'bounce' an object will have. There shouldn't be bounce in the rollers of the Mecanum wheels. Bounce would let some rollers float in the air instead of sliding over the surface, at the expense of precision.

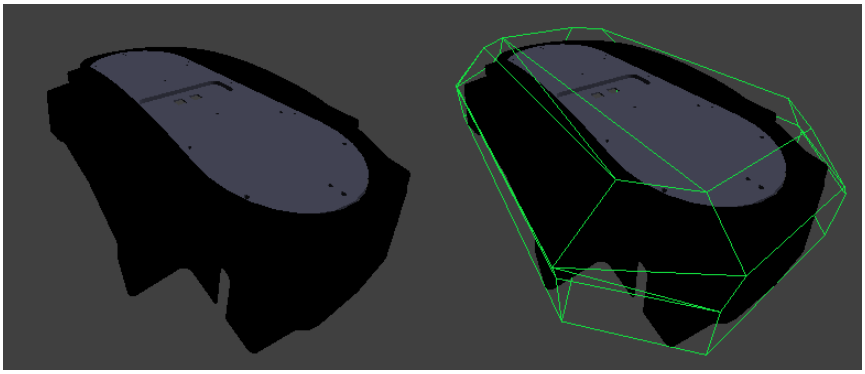

Figure 3.4: Left the static mesh, right the static mesh with collision

3.5 Design of the KUKA youBot in USARSim

In order to simulate the movement the KUKA youBot base in a highly accurate way only using rigid body physics is not enough. Our simulated robot should also have its physical material defined, more precisely physical material should be applied on the rollers of the Mecanum wheels, because the only contact points with the ground are made by rollers of the Swedish wheels.

The values for the physical material described in the previous section are left to the default value except for friction. Further experimenting with these values could definitely bring more precision in the movement behavior of the Mecanum wheels.

In Figure (3.5) an abstract picture of the KUKA youBot base is shown. The blue elements are the body and the four main wheels of the base, which are rigid bodies. The red elements are the rollers of the Mecanum wheels, which are rigid bodies with physical material applied to them.

The idea to design the KUKA youBot base is complete. The body should be a rigid body, including the static mesh and physics. The four main wheels should also be rigid bodies, including the static mesh and physics. The connection between the body and the main wheel should be joints. This implementation will use the wheeljoint class to specify the joint for each of the main wheels. The child of the joints will be a main wheel and the parent will be the KUKA youBot base. The rollers should be rigid bodies including physical material. There are six roller per main wheel. These twenty-four rollers are linked to the main wheel with rollerjoints. The rollerjoint

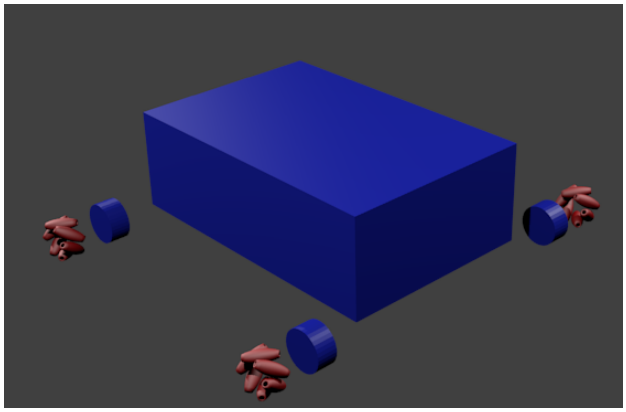

Figure 3.5: Abstract representation of the KUKA youBot base in Blender

class is a custom made wheeljoint class, where specific joint values may differ. The child of the roller joints will be the rollers and the parent will be their respective main wheel. Maximum velocity and stiffness are two parameters of the roller joints that potentially have an impact on the magnitude of the resulting movement. They were left to the default values in the design but further optimization is a possibility.

The robot class is defined as the KUKA youBot class, which can be found in the appendix. This class extends the custom made 'omnidirectional steered' class, which listens to the controller and parses the 'DRIVE' command into four wheel speeds. The command for driving all four wheels with speed 1 has the form 'DRIVE {Front-Left 1,00}{Front-Right 1,00}{Back-Left 1,00}{Back-Right 1,00}'. The wheel speeds are assigned to the joints with the sides front-left, front-right, back-left and back-right.

CHAPTER 4

KUKA youBot

The term robot was first introduced in 1921 by the Czech playwright Karel Čapek in his play R.U.R. [28]. The word 'robota' is the Czech word for work. Since then the term has been applied to a great variety of mechanical devices. The official definition of the term robot is given by the Robot Institute of America[1] as: "A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices trough variable programmed motions for the performance of a variety of tasks". The most important part of this definition in the field of computer science is the reprogrammability of the robot, making the robot adaptable for a wide variety of tasks.

The KUKA youBot as seen in Figure (4.1), is a mobile robot developed as open source platform for scientific research and education. The robot consist of a mobile platform with four Mecanum wheels and two optional robot arms. These components are driven by motors of Maxon Motor[2]. Two years after the KUKA youBot was introduced to the market, the KUKA has had a significant impact on research and education on mobile manipulators. KUKA is known for its industrial robots, however they produced the KUKA youBot to promote the research in robotics [26]. The KUKA youBot is a open source platform created for robotic research, in which scientists, developers en students can write their own software for the robot, e.g. mounting tables or transportation of supplies.

Figure 4.1: The KUKA youBot

The youBot comes with fully open interfaces and allows developers to access the system on nearly all levels of hardware control. It has an application programming interface, with interfaces and wrappers for current robotic frameworks like ROS[3] or OROCOS[4]. The high level of access in the robot system lets the developer control each actuator and sensors directly. The

only limitations are the youBot's kinematic structure, its drive electronics and its motors. The KUKA youBot qualifies as a so-called partly completed machinery, it does not come with any preinstalled software. The software is mostly open source and can be found on KUKA youBot store.[5]

The KUKA youBot consist of an omnidirectional mobile platform with an one or two optional robot arms mounted on top of the chassis. In the chassis an industrial PC and a battery are integrated. The industrial PC communicates in real time with a time-step of 1 ms, through a EtherCAT with a total of nine motors, used for regulation of current, speed and position.

The chassis of the robot consist of a platform of 53 cm by 36 cm with a height of 11 cm, with four Mecanum wheels. On the chassis a 66 cm long arm is mounted with a gripper consisting of two fingers. The gripper can grab objects with a size of 7 cm with a weight of up to 500 g. The robot arm has five joints.

The KUKA youBot can be seen as a milestone in research and development in robotics.

[1]Robot Institute of America, 1979
[2]http://http://www.maxonmotor.com/
[3]http://www.ros.org/
[4]http://www.orocos.org/
[5]http://www.youbot-store.com/

CHAPTER 5

Mecanum wheels

The Mecanum wheel is also called the Ilon wheel after its Swedish inventor, Bengt Ilon[1]. Ilon invented the wheel in 1973 while working for the Swedish company Mecanum AB. The wheel consists of a main wheel and rollers mounted around its periphery. The rollers mounted on the main wheel can have any angle in relation to the main wheel. In case of the conventional Swedish wheel this angle is 45 degrees. The wheels have the feature to provide the vehicle omnidirectional movement. This means the vehicle can move in any direction of the plane, without the requirement of making rotation. The scope of this thesis is to describe the movement of the vehicle, which can be broken down in the movement of the four Mecanum wheels. Movement will be described by mobility, kinematics, and dynamics.

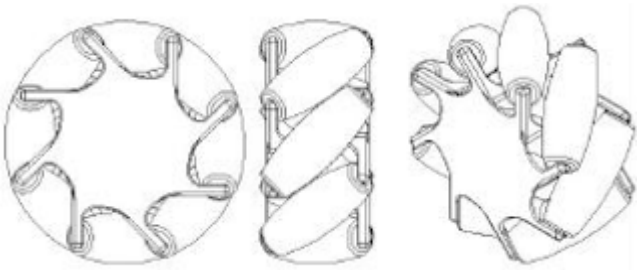

Figure 5.1: Schematic of the Mecanum wheel [27]

Mecanum wheels are used by several robots. There are many variations possible, e.g. the number of rollers or the angle of the rollers. One improved design is splitting each roller into two [27].

5.1 Mobility

The kinematic mobility of a robot chassis is its ability to directly move in the environment. Mobility is a definition covering several movement aspects. In describing the mobility of a robot chassis movement aspects are a degree of maneuverability, a degree of steerability and degrees of mobility. The degree of steerability is the number of controllable steerable axes. A configuration of a robot is a complete specification of the possible movements. An object is said to have a degree of freedom of n if its configuration can be minimally specified by n parameters.

The overall degrees of freedom (DOF) that a robot can manipulate is called the degree of maneuverability δ_M. This is composed by the degree of mobility and the degree of steerability with the equation [13]

$$\delta_M = \delta_m + \delta_s \tag{5.1}$$

[1] http://www.google.com/patents/US3876255

The Mecanum wheel has a DOF of three, indicated with arrows in Figure (5.2). The first DOF is in the direction of the wheel rotation. The second DOF is provided by motion of the rollers mounted at a 45 degrees angle of the main wheel. In the Figure (5.2) the roller on top of the wheel is shown, the roller at the bottom provides the movement, which is perpendicular to the roller on top of the wheel. The third DOF is the rotational slip at the point of contact with the floor. In the case of the KUKA youBot the wheels are also not steerable, providing the robot a steerability of zero.

Thus the degree of maneuverability for our robot given Equation (5.1) is three. This does not mean that our chassis is capable of 3D movement, it provides omnidirectional movement for a (x,y) plane.

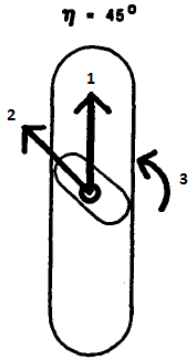

Figure 5.2: Three DOF of the Mecanum wheel [1]

5.2 Movement

In Figure (5.3) the required motions of the wheels for side movement of the robot is shown. The combination of the left-front-wheel moving backwards and the left-back-wheel moving forward results in a movement to the left. Also the combination of the front-right-wheel moving forward and the back-right-wheel moving backwards results in a movement the to left. This is because the forwards and backwards forces of the main wheels are balanced out and the only forces left are due to the rollers of the Mecanum wheels. Note that in this top view it may seem that the DOF of the roller is along its long side, this is not the case because the roller on the bottom is pivoted 180 degrees and the DOF is perpendicular to its long side.

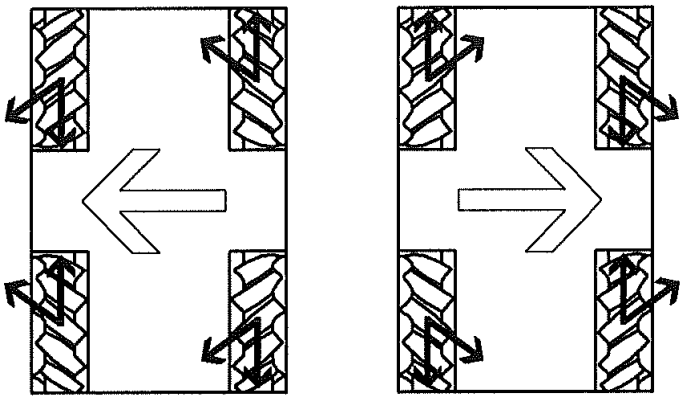

Figure 5.3: Top view of side movement of the robot [1]

[1] https://www.google.com/patents/US20130068543

CHAPTER 6

Kinematics

Kinematics describes the motion of a robot without any consideration of the forces or the torques causing this motion. The kinematics of the KUKA youBot base can be decomposed into the kinematics of the four Mecanum wheels. Kinematics can be divided into forward kinematics and inverse kinematics. Forward kinematics describes how to determine the position and orientation of a robot given the values of the joint variable of the robot. Inverse kinematics describes values of the joint variable of the robot given the position and orientation of the robot. The goal is to make an inverse kinematic model for the KUKA youBot base. This model consists of a set of matrices, that will result in the position that the four wheels have to be, given the position and orientation of the robot. The four wheels each have their own frame and should be transformed using transformation matrices. A frame is a space with its own unit vectors and origin. The robot also has a frame, which will be the main frame in the final equation. Firstly it will be described how to work with different frames by introducing transformation matrices. Then the coordinate frames of the robot will be defined. Finally its corresponding transformation matrices used in the inverse kinematic equation of the KUKA youBot base will be the result.

6.1 Transformation matrices

Transformation matrices grant the ability to work in different frames by allowing redefinition of vectors from one coordinate frame onto another. Transformation of a vector is divided into rotation and translation. The notation $\begin{pmatrix} a & b \end{pmatrix}^T$, could be a 2D point in a coordinate frame with a and b as scalar values or it could be a vector with a direction and a magnitude. In this section this notation will be used to denote a vector with a direction and a magnitude. When transforming rotation of a vector, the vector is not bound to be located at a particular point in space. Only direction and magnitude matter when transforming the rotation of a vector into another frame. It is not necessary to transform the rotation of a vector when both coordinate frames are parallel. However when working with non-parallel frames, it is required to use a rotation matrix to express the vector from one coordinate frame onto another.

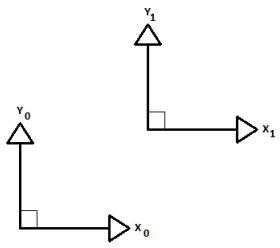

Figure 6.1: Rotation between parallel coordinate frames is not necessary.

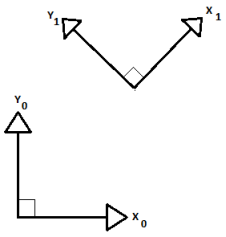

Figure 6.2: Rotation between non parallel coordinate frames is necessary.

The notation for rotation matrix R_1^0 in a three-dimensional space with origin o and with axes (x, y, z) for transforming free vectors from frame $o_0x_0y_0z_0$ to frame $o_1x_1y_1z_1$ will be used in this section [22].

$$R_1^0 = \begin{pmatrix} x_1^0 & y_1^0 & z_1^0 \end{pmatrix} \tag{6.1}$$

In Equation (6.1), x_1^0 is the vector from frame $o_1x_1y_1z_1$ expressed in what they would be in frame $o_0x_0y_0z_0$. This mean that the x-component from frame $o_1x_1y_1z_1$ can be projected on the axes of frame $o_0x_0y_0z_0$. The dot product of two vectors gives the projection of one vector onto the other. Resulting in the following equation

$$x_1^0 = \begin{pmatrix} x_1 \cdot x_0 \\ x_1 \cdot y_0 \\ x_1 \cdot z_0 \end{pmatrix} \tag{6.2}$$

Combining Equations (6.1) and (6.2), the resulting rotation matrix is given by

$$R_1^0 = \begin{pmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 & z_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot y_0 & z_1 \cdot y_0 \\ x_1 \cdot z_0 & y_1 \cdot z_0 & z_1 \cdot z_0 \end{pmatrix} \tag{6.3}$$

This rotation matrix can be used to transform vectors expressed in frame $o_1x_1y_1z_1$, into vectors expressed in frame $o_0x_0y_0z_0$. This matrix can be used to transform vector p_1 from frame $o_1x_1y_1z_1$ into p_0 to his corresponding length and magnitude expressed in frame $o_0x_0y_0z_0$.

$$p^0 = R_1^0 p_1 \tag{6.4}$$

Rotation matrices can be combined with each other with the following properties

$$R_2^0 = R_1^0 R_2^1 \tag{6.5}$$

The use of Equation (6.5) transforms a vector given in coordinates of frame $o_2x_2y_2z_2$ to coordinates of frame $o_0x_0y_0z_0$, which results in the following formula

$$p^0 = R_1^0 R_2^1 p^2 \tag{6.6}$$

This is the law for rotational transformations. The order of the multiplication of the rotation matrices is crucial [23]. Similarity transformations are transformations between two frames with the product being a linear transformation instead of a vector. A coordinate frame is defined by a set of basis vectors. A rotation matrix can be viewed as redefining these basis vectors. The matrix representation of a general linear transformation is transformed from one frame to another using a similarity transformation. If A is the matrix representation of a given linear transformation in frame $o_0x_0y_0z_0$ and B is the representation of the same linear transformation in frame $o_1x_1y_1z_1$, then A and B are related as

$$B = (R_1^0)^{-1} A R_1^0 \tag{6.7}$$

Not only rotation is necessary within a transformation matrix but also translation. The use of homogeneous coordinates allows the combination of rotation and translation in a matrix. The homogeneous transformation matrix has the form

$$H = \begin{pmatrix} R & d \\ 0 & 1 \end{pmatrix} \tag{6.8}$$

with the rotation matrix R and the translation vector d

$$d = \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} \tag{6.9}$$

and combines rotation and translation in one matrix. The d-component of the matrix defines the translation in the (x, y, z) axes. The previously stated rules for the rotation matrices also apply

to the homogeneous matrices. The homogeneous matrices can be used to transform vectors or transformations from one frame to another. The possibility to combine the vectors of different frames into one result frame grants the ability to work with several frames when describing the kinematics of the KUKA youBot base.

6.2 Coordinate frames

The robot frame is defined at the center of the KUKA youBot base; four contact frames at wheel axles of the Mecanum wheels and the world frame at a fixed location. The robot frame and the contact frames are assigned parallel. Therefore the transformation matrices of the contact frames to the robot frame define translation and no rotation.

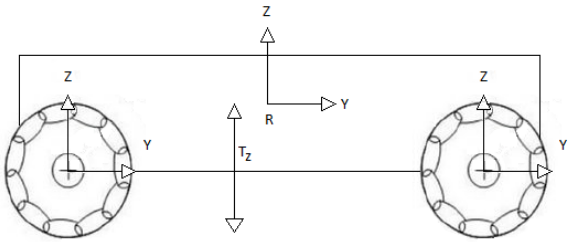

Figure 6.3: Schematic side-view of KUKA youBot with X axis pointing out the page

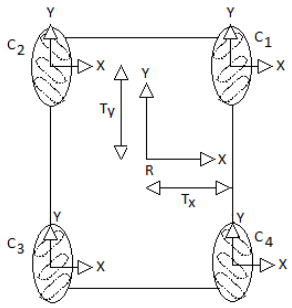

Figure 6.4: Schematic top-view of KUKA youBot with Z axis pointing out the page

The T_x, Ty and T_z vectors in Figure (6.3) and (6.4) denote the distance between the robot frame and the contact frames in the x, y and z directions. The homogeneous transformation matrices for transforming vectors from the contact frames C_1, C_2, C_3 and C_4 into the robot frame R, as seen in Figure (6.4) are given by

$$\Pi^R_{C_1} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \Pi^R_{C_2} = \begin{pmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.10}$$

$$\Pi^R_{C_3} = \begin{pmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \Pi^R_{C_4} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.11}$$

6.3 Jacobian matrix

A Jacobian matrix defines the velocity relationships within a certain frame. The coordinate frames of the robot and the transformation matrices for working with different frames are specified

in the previous sections. Now the focus will be on how to represent velocities in the wheels of the robot base. There are two possible velocities, angular velocity and linear velocity. Both can be in the same moment in a frame, just like translation and rotation. With the use of Jacobian matrices the velocity relationships can be derived for the Mecanum wheels, relating the linear and angular velocities of the end effector to the wheel velocities. Linear velocity is the velocity of a rigid body in a straight line. When a rigid body is moving with constant speed without changing its direction then it is said to be moving with the constant linear velocity. Linear velocity signifies that direction is not changing, while constant velocity signifies that the magnitude of the velocity is not changing. Pure angular velocity is the velocity of an object purely rotating around a certain axis. When a rigid body turns or moves in the circular direction, it has both linear velocity and angular velocity. The angular velocity of a particle moving on a circular path, is defined as the ratio of the angle traversed to the amount of time it takes to traverse that angle. The Jacobian is a matrix that generalizes the notion of the ordinary derivative of a scalar function.

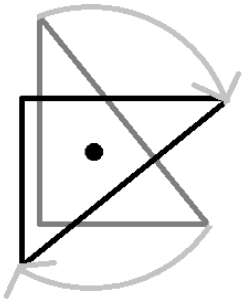

Figure 6.5: Angular velocity of a triangle

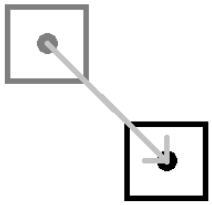

Figure 6.6: Linear velocity of a square

When a rigid body purely rotates over an angle θ, meaning there is no translation of the body, any point of the body rotates over θ. If k is a unit vector in the direction of the axis of rotation, then the angular velocity is given by

$$\omega = \dot{\theta}k \tag{6.12}$$

in which $\dot{\theta}$ is the time derivative of θ.

The linear velocity of any point on the body is given by the equation

$$v = \omega \times r \tag{6.13}$$

in which r is a vector from the origin to the point.

Looking at the motion of a moving frame, angular velocity is in our interest. The goal is to compute relative velocity transformations between coordinate frames.

6.3.1 Derivative of the Jacobian

We introduce skew symmetric matrices to simplify the equations made for the Jacobian matrix. A $n \times n$ matrix S is said to be skew symmetric if and only if the following formula applies

$$S + S^T = 0 \tag{6.14}$$

A 3×3 skew matrix will be denoted as $so(3)$.

If $S \in so(3)$ has component s_{ij}, with $i, j = (1, 2, 3)$, then $s_{ij} + s_{ji} = 0$, for i, j in $(1, 2, 3)$.

Any vector can be rewritten in skew symmetric matrix form. For the vector $a = \begin{pmatrix} a_x & a_y & a_z \end{pmatrix}$, the skew matrix of the vector is defined as

$$S(a) = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix} \tag{6.15}$$

Suppose that rotation matrix R is a function of rotation value θ. Hence, $R = R(\theta) \in SO(3)$ for every θ. Given $R(\theta) \in SO(3)$, R is orthogonal for all θ, the following rule applies for the rotation matrix

$$R(\theta)R(\theta)^T = I \tag{6.16}$$

Differentiating both sides in Equation (6.16) on θ, the product rule provides the following formula

$$\frac{\mathrm{d}}{\mathrm{d}\theta}RR(\theta)^T + R(\theta)\frac{\mathrm{d}}{\mathrm{d}\theta}R^T = 0 \tag{6.17}$$

Let us define S as

$$S = \frac{\mathrm{d}}{\mathrm{d}\theta}RR(\theta)^T \tag{6.18}$$

with the transpose

$$S^T = R(\theta)\frac{\mathrm{d}}{\mathrm{d}\theta}R^T \tag{6.19}$$

Given Equation (6.17) and the definition of S and S^T we can say that $S + S^T = 0$, making $\frac{\mathrm{d}}{\mathrm{d}\theta}R(\theta)^T$ a symmetric skew matrix. Multiplying both sides of Equation (6.18) by R and using $R^T R = I$ the following formula results

$$\frac{\mathrm{d}}{\mathrm{d}\theta}R = SR(\theta) \tag{6.20}$$

This formula is the definition of the derivative of the rotation matrix.

The Jacobian allows the expression of angular velocity and linear velocity to a given point. The rotation matrix R is time varying, written as R(t). As already shown in Equation (6.20) the derivative of a rotation matrix R(t) can be written as

$$\dot{\mathrm{R}}(t) = S(\omega(t))R(t) \tag{6.21}$$

where $S(\omega(t))$ is a skew symmetric matrix and ω is the angular velocity with respect to the frame. To use this formula for two different coordinate frame the next formula is given

$$\begin{aligned} p^0 &= R_1^0 p^1 \\ \frac{d}{dt}p^0 &= \dot{\mathrm{R}}_1^0 p^1 \\ &= S(\omega)R_1^0 p^1 \\ &= \omega \times R_1^0 p^1 \\ &= \omega \times p^0 \end{aligned} \tag{6.22}$$

The step from is using Equation (6.21). The step from uses the fact that $S(a)p = a \times p$. Angular velocities from several different coordinate frames can be added up. As previously stated in Equation (6.5) rotation matrices can be combined. The derivative of this results in

$$\dot{\mathrm{R}}_2^0 = \dot{\mathrm{R}}_1^0 R_2^1 + R_0^1 \dot{\mathrm{R}}_2^1 \tag{6.23}$$

All the elements of this equation can be decomposed by the following equations

$$\dot{\mathrm{R}}_2^0 = S(\omega_{0,2}^0)R_2^0 \tag{6.24}$$

With $R_2^0 = R_1^0 R_2^1$

$$\dot{\mathrm{R}}_1^0 R_2^1 = S(\omega_{0,1}^0)R_2^0 \tag{6.25}$$

$$R_0^1 \dot{\mathrm{R}}_2^1 = R_1^0 S(\omega_{1,2}^1)R_2^1 = S(R_1^0 \omega_{1,2}^1)R_2^0 \tag{6.26}$$

As a result every component has R_2^0 and since S(a) + S(b) = S(a+b)

$$\omega_2^0 = \omega_{0,1}^0 + R_1^0 \omega_{1,2}^1 \tag{6.27}$$

Angular velocity and linear velocity are combined by use of homogeneous matrix. The matrix is a function of time and is defined as

$$H_1^0(t) = \begin{pmatrix} R_1^0(t) & o_1^0(t) \\ 0 & 1 \end{pmatrix} \tag{6.28}$$

where $R_1^0(t)$ handles the angular velocity by expressing the speed at which the frame is rotating and $o_1^0(t)$ handles the linear velocity by expressing the speed at which the origin of the frame is moving.

6.3.2 Jacobian for the KUKA youBot

The Jacobian matrix of the KUKA youBot base is made up out of four Jacobian matrices of the four contact frames, positioned at the axis of each wheel. The Jacobian matrix for contact frame i is defined as:

$$J_{C_i} = \begin{pmatrix} -R_i sin\theta_{C_i}^R & r_i sin(\theta_{C_i}^R + \eta_i) & -d_{C_i y}^R \\ R_i cos\theta_{C_i}^R & -r_i cos(\theta_{C_i}^R + \eta_i) & -d_{C_i x}^R \\ 0 & 0 & 1 \end{pmatrix} \tag{6.29}$$

Where R is the circumference of main wheel i, r is the circumference of roller of wheel i. To further simplify this equation, the wheels and the rollers all have the same circumference, so $R_1 = R_2 = R_3 = R_4 = R$ and $r_1 = r_2 = r_3 = r_4 = r$. The magnitude of the roller angles are also equal for wheel 1 and 3, and wheel 2 and 4 have this angle reversed. The Mecanum wheels of the KUKA youBot are the traditional Swedish wheels, so $\eta_1 = \eta_3 = -45°$ and $\eta_2 = \eta_4 = 45°$. The Jacobian matrices for the four wheels are defined as:

$$J_{C_1} = \begin{pmatrix} -Rsin(\theta) & rsin(\theta - 45) & T_y \\ Rcos(\theta) & -rcos(\theta - 45) & T_x \\ 0 & 0 & 1 \end{pmatrix} \tag{6.30}$$

$$J_{C_2} = \begin{pmatrix} -Rsin(\theta) & rsin(\theta + 45) & T_y \\ Rcos(\theta) & -rcos(\theta + 45) & -T_x \\ 0 & 0 & 1 \end{pmatrix} \tag{6.31}$$

$$J_{C_3} = \begin{pmatrix} -Rsin(\theta) & rsin(\theta - 45) & -T_y \\ Rcos(\theta) & -rcos(\theta - 45) & -T_x \\ 0 & 0 & 1 \end{pmatrix} \tag{6.32}$$

$$J_{C_4} = \begin{pmatrix} -Rsin(\theta) & rsin(\theta + 45) & -T_y \\ Rcos(\theta) & -rcos(\theta + 45) & T_x \\ 0 & 0 & 1 \end{pmatrix} \tag{6.33}$$

For example if wheel 1 has been assigned a speed of 1 in the y-direction the equation would be

$$\begin{pmatrix} 0 & -r\sqrt{(}2)/2 & T_y \\ R & -r\sqrt{(}2)/2 & T_x \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -r\sqrt{(}2)/2 \\ -r\sqrt{(}2)/2 \\ 0 \end{pmatrix} \tag{6.34}$$

6.4 Composite youBot base equation

The motion of the KUKA youBot base is the result of the movement of the four Mecanum wheels. The motion for the Mecanum wheel is provided as a Jacobian matrix. The Jacobian matrices can be combined trough the following equation

$$J_{Robot} = \begin{pmatrix} J_{C_1} & 0 & 0 & 0 \\ 0 & J_{C_2}0 & 0 & 0 \\ 0 & 0 & J_{C_3} & 0 \\ 0 & 0 & 0 & J_{C_4} \end{pmatrix} \tag{6.35}$$

The robot matrix is a (12×3) matrix filled with zeros, where J_{C_1},J_{C_2},J_{C_3} and J_{C_4} are the previously defined Jacobian matrices for the wheels. The matrix has as input the composite wheel velocity vector, defined as

$$\begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{pmatrix} \tag{6.36}$$

With $\dot{q}_i$ is the velocity vector with (x,y,z)-components for Mecanum wheel i.

In summary, the previously defined robot and wheel frames and the Jacobian matrix are combined as shown in Figure (6.7). The roller angle of -45 degrees causes the second movement. Various angles of the rollers are possible, 45 degrees provides the maximum amount of side-way movement whereas 0 degrees provides none. This model lacks any notion of friction. Therefore it is not complete, however, it should provide a basis for understanding the motion of the wheels.

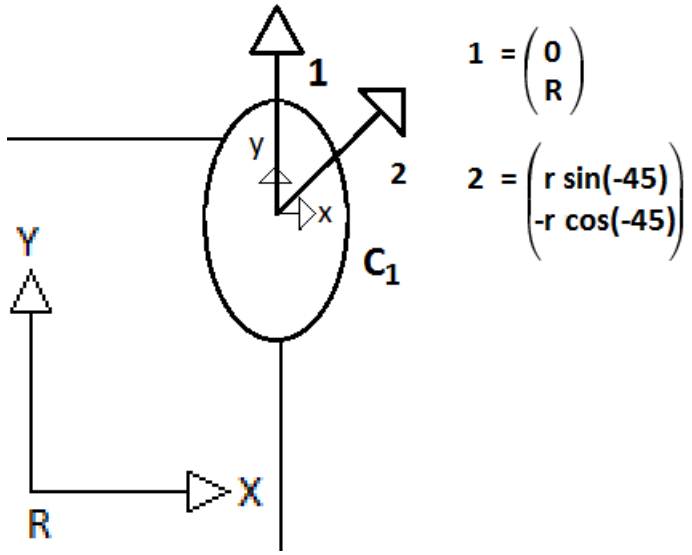

Figure 6.7: Two movement speeds of wheel 1

CHAPTER 7

Experiments

The experiments provide the ability to compare movement behavior between the real KUKA youBot and the simulated implementations. There are four types of experiments, each will be executed five times.

- The first experiment is to move forward ten steps.
- The second experiment is to move backward ten steps.
- The third experiment is to move left ten steps.
- The fourth experiment is to move right ten steps.

All the four experiments are executed on a rough surface. Also the experiments are repeated on a smooth surface, which has the goal to provide results with a different friction. The first and the third experiment will also be executed with a small elevation element. For the first movement, the robot will drive up a small slope. The elevation experiment will be repeated for two different elevation settings. All these experiments will be performed by the real KUKA youBot, the simulation approximation of the robot and the kinematic simulation of it. The goal for the experiments is to provide validation of the movement of both simulated implementations. A friction and an elevation element are added to further compare the movement.

7.1 Set-up

The real KUKA youBot was steered using the ROS Wrapper for the KUKA youBot API. The real KUKA drove after the user hit a button linked to the movement command. Subsequently the measurement was preformed and the next command was executed. The rough surface was created using synthetic carpet used in RoboCup 2013 Eindhoven. Also the elevation element was covered with this carpet. The angle of the slopes were 1.5 degrees for the first elevation and 4.5 degrees for the second. The smooth surface was provided by a cast acrylic sheet 'ISO9001'.

The set-up for the virtual experiment in USARSim was using the standard Iridium controller. The Iridium code was extended with an experiment section which gave DRIVE commands to the robot after it was initiated. Interference of the user was not needed. These commands were separated with the sleep function of Java. For this experiment three maps were used in the Unreal engine, one with a rough surface, one with a smooth surface and one with a rough surface with an elevation element. The rough surface was made with physical material with the default values. And the smooth surface was made by physical material with glass as material type. For the elevation element gravity is an important aspect in the simulation, the gravity of USARSim is validated [9].

The code for the real KUKA youBot and the simulated KUKA youBot are the same for the vertical movement. The real KUKA youBot was steered with equivalent `C++` code steered by the ROS Wrapper for the KUKA KUKA youBot API. However the horizontal movement of the ROS

Wrapper seemed to be scaled up to have both vertical and horizontal movement match. This was noted after the results, therefore the results of the real KUKA youBot's horizontal movement are scaled with the formula

$$horizontalmovement_{real} = horizontalmovement_{real} \times \frac{\sqrt{2}}{2}$$

This formula was obtained by the y-movement of the previously made Jacobian matrix.

7.2 Sensor

In the real experiments measuring tape was used to determine the position of the robot. A reference point was set at the robot and two tapes were used to provide a measuring grid.

In USARSim the groundtruth sensor was used to log the position of the robot. The groundtruth sensor has a frequency of ten measurements per second. These values were handled by receiving the USARpackets with the Iridium controller, parsing the location and values and storing these in a text file. The real youBot was measured once per movement and to adjust to this the abundant sensor information of the groundtruth sensor was filtered at the first measurement per twenty measurements. Twenty measurements were the time that one move was executed.

7.3 Parameter optimization

Before the real experiments are conducted, parameters should be optimized. This is done by comparing one experiment of the real robot to various simulated experiments with different parameters. The effect of friction of the rollers and weight of the chassis of the KUKA youBot are tested.

7.3.1 Effect of friction on error in horizontal movement

The physical material of the rollers has a friction value. To determine the optimal value for friction, experiments were done with friction values ranging from 0.50 to 1 with steps of 0.1. The comparison was done with the real robot and the simulated one on the experiment to drive right on the rough surface. This process was repeated five times for each friction value and the results are shown in Figure (7.1). The movement to the right was very similar for all friction values. However, the difference in forward movement differed for every value.

It is clear that the friction value of 1.00 has the least amount of error, however the early rise in error followed by the decline to zero is suspicious. Friction values 0.60 and 0.70 show a error in the backward-direction and friction values 0.50, 0.80 and 0.90 show a error in the forward-direction.

In Figure (7.2) the forward movement of the real robot in shown for five samples. It is clear that the real robot has an error in the forward direction.

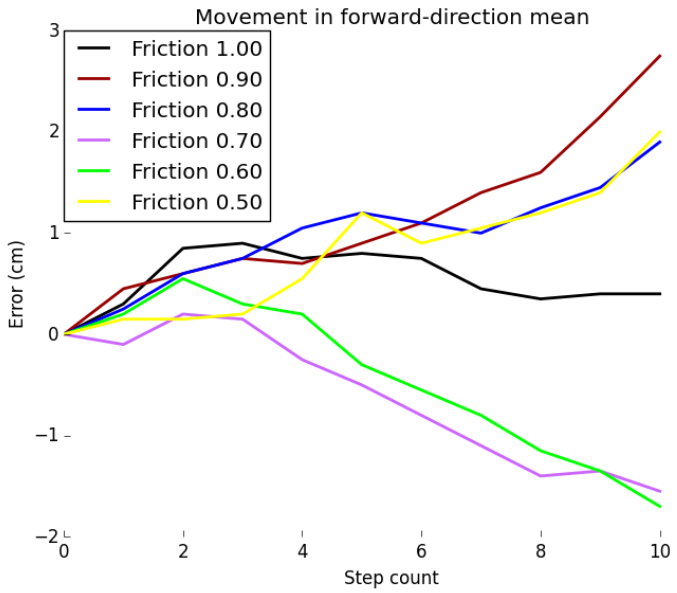

Figure 7.1: Mean difference in forward-direction over step count for all friction values

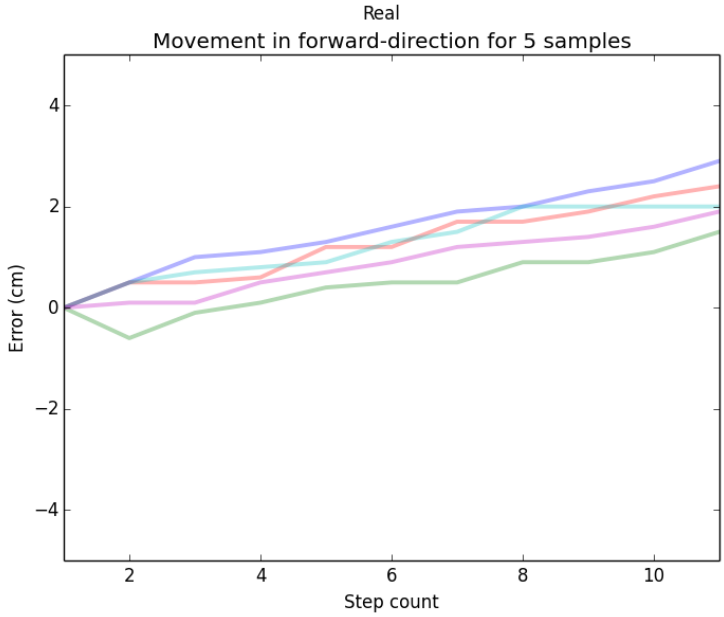

Figure 7.2: Difference in forward-direction over step count for five samples

After examining friction values 0.50, 0.80 and 0.90 and 1.00 the value 0.90 seemed best, shown in Figure (7.3) None of the five samples had a clear movement in the backward-direction, this was the case for all the other values causing there mean to be less. It is worth noting that the variance in the forward-direction when moving to the right is a lot higher for the simulation than for the real robot. The results for the other friction values can be seen in the appendix.

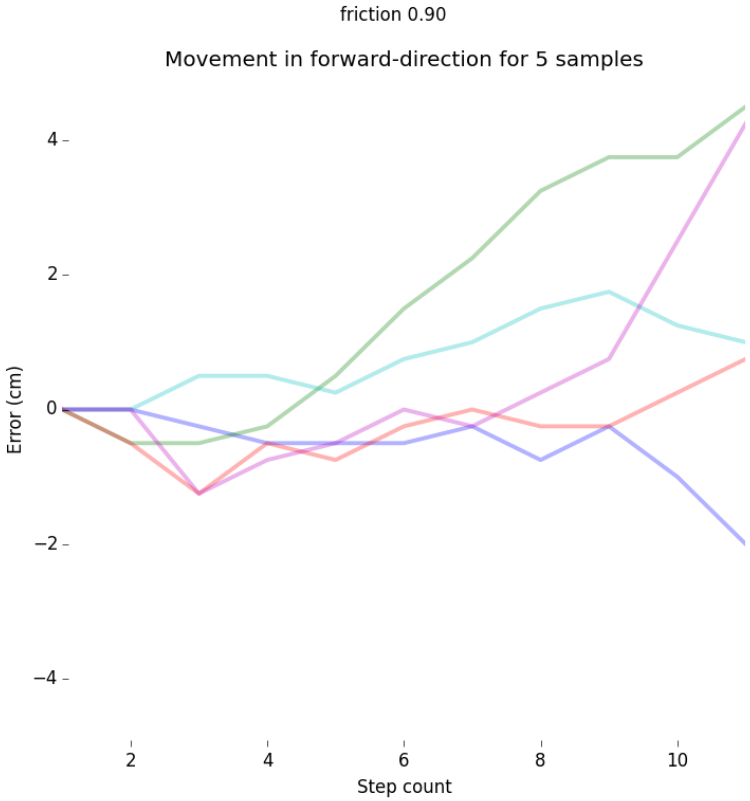

Figure 7.3: Difference in forward-direction over step count for five samples

7.3.2 Effect of weight on step size

The weight of the KUKA youBot chassis could affect the step size in horizontal movement. The experiment to the right on the rough surface was done for weight values of 12,16 and 20 kg. The effect of the step size is shown in Figure (7.4). All lines are plotted on top of each other. This means that different weight does not have a significant effect on the step size in horizontal movement.

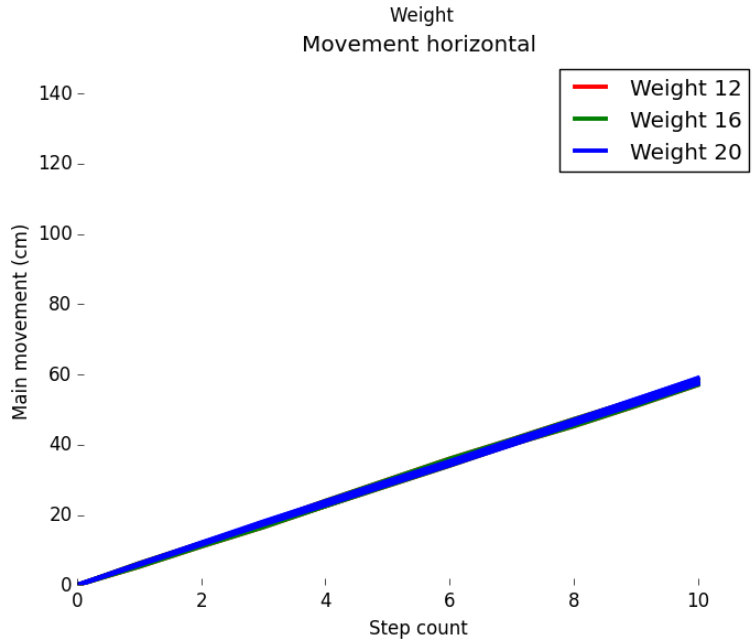

Figure 7.4: Difference in movement

7.4 Results

In this section the results of the experiments comparing the real KUKA youBot to its simulated version are provided. The movements forward and left are used to represent the vertical and horizontal motions. A vertical motion has a step size in the vertical axis and an error in the horizontal axis. A horizontal motion has a step size in the horizontal axis and an error in the vertical axis. The results are presented for five samples per experiment. Since error can be in the positive- or the negative-direction no mean was given. A positive and a negative error even each other out in the mean value. Only plotting absolute error values does not provide the direction of the error, which can be useful in interpreting the data. Lines are chosen to visualize in a clearer way, this does not mean the data can be interpolated between points. The step count is always ten steps, so only ten data points should be interpreted.

7.4.1 Average step size

In the Figures (7.5) and (7.6) the average step size is shown. In Figure (7.5) the average step in the main movement is shown. The Figure shows that the main movement for every simulated experiment was smaller than the real one. In Figure (7.6) the average step in the error-direction is shown. The error step size is not direction dependent, only the difference in each step was measured. Smooth forward for the simulation shows a very large error. This is due an exponential error increase, this indicates rotation of the simulated vehicle.

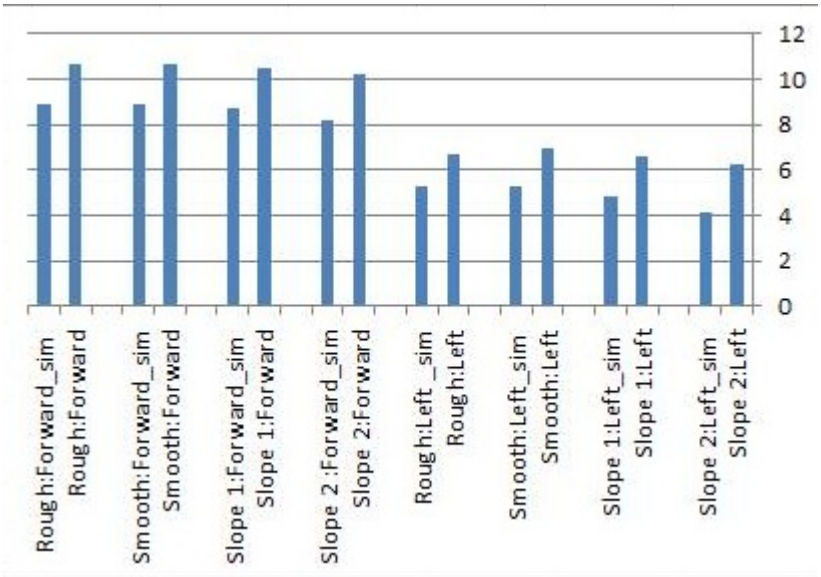

Figure 7.5: Average step size in main movement

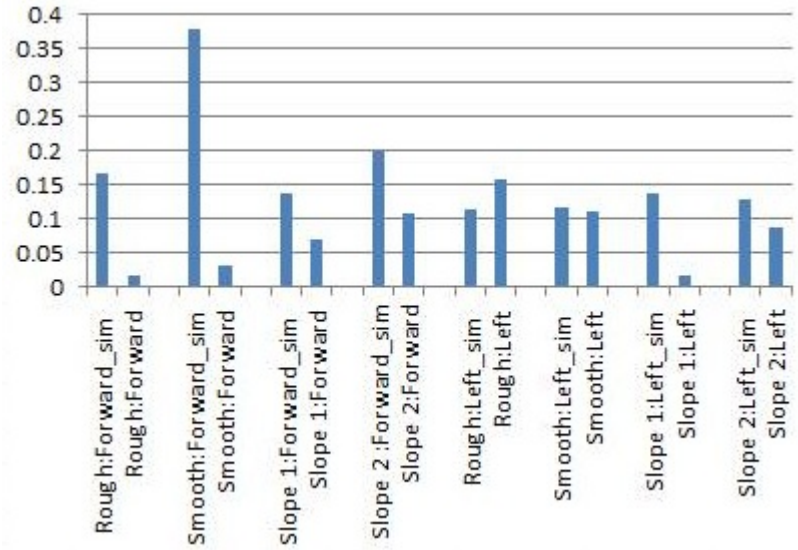

Figure 7.6: Average step size in error-direction size

7.4.2 Forward movement for different surfaces

The forward movement for the rough and smooth surface is presented in Figure (**??**). In the figure the movement in the y-direction (error) for the forward movement is shown. The variance in the y-direction is higher for the smooth surface compared to the rough surface in the y-direction for the real robot. The error for the real robot is linear. The variance for the simulated compared to the real robot is considerably higher and seems to favor the left-direction. The error for the simulated robot is exponential which indicates a rotation.

The forward movement step size is closely resembled with a consistent smaller step size for the simulated version. Also the change of surface friction did not effect the step size for both the simulated as well as the real robot. This resemblance is shown in the appendix. Overall the main movement for the real robot was about 108 cm in ten steps and about 98 cm for the simulated one. The error for the real robot was about 1.0 cm and did not favor any direction. The error for the simulated robot was exponential and favored the left-direction, indicating a small left rotation in the vehicle.

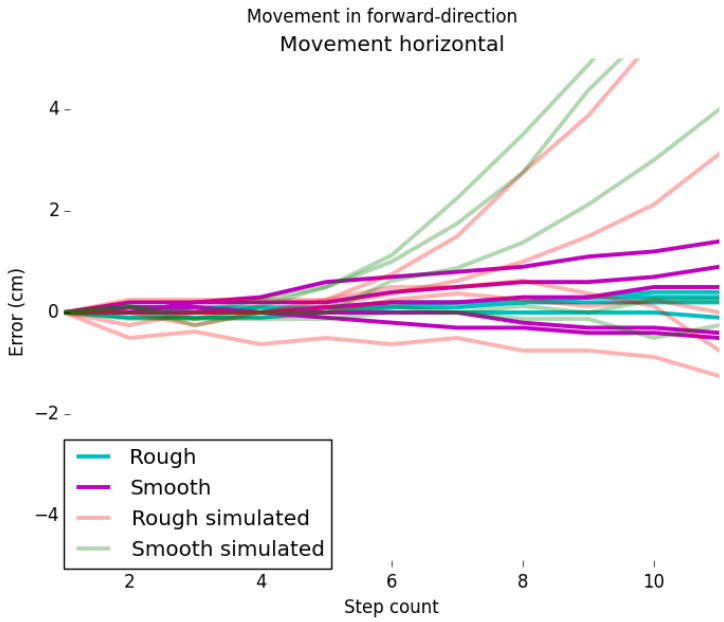

Figure 7.7: Movement forward on the rough surface, difference in y-direction plotted

7.4.3 Forward movement for different elevations

The step size for the forward movement on rough surface for 0, 1.5 and 4,5 degrees as "Rough", "Slope 1" and "Slope 2 " is shown in Figure (7.8). For the real robot the step size is lower as elevation is higher. The difference between 0 and 1.5 is 1 cm and the difference between 1.5 and 4.5 is 2 cm after ten steps. This indicates that a 1.5 degrees in angle for the surface causes a linear decrease of 0.1 cm in step size. For the simulated robot the step size seems more effected by elevation. As the difference between 0 and 1.5 is 1.5 and the difference between 1.5 and 4.5 is 6 cm after ten steps. This indicates that the error is exponential. More elevation should be tested to give a good estimate of the exponential function.

In Figure (7.9) the error for the forward movement on the different elevations is shown. The real robot's error is mostly in the left-direction and is linear. The error for the simulated robot is also mostly in the left-direction but is more spread. The error also is exponential for numerous results, mostly at the 4.5 degrees angle. This angle also produces a error mostly in the right-direction for the simulated robot whereas for lower angles this was mostly the left-direction.

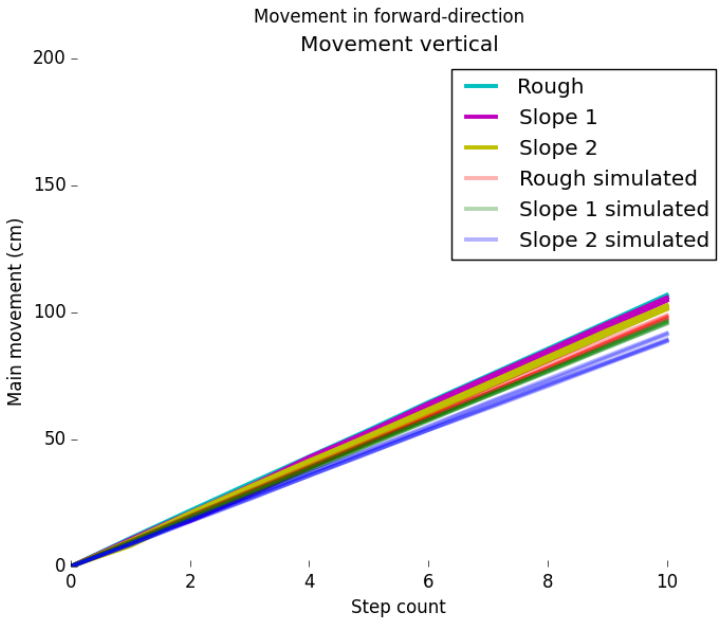

Figure 7.8: Movement forward on the rough surface, difference in x-direction plotted

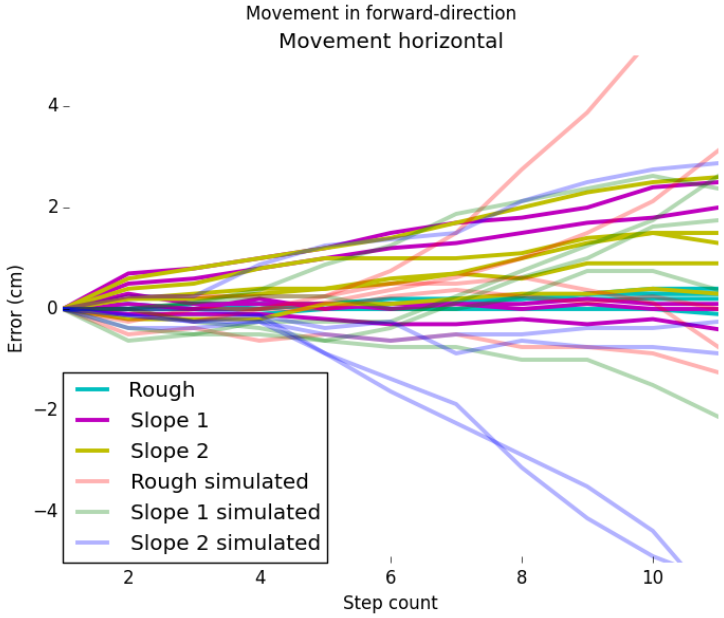

Figure 7.9: Movement forward on the rough surface, difference in y-direction plotted

7.5 Left movement for different surfaces

The left movement for the vehicle on the rough and smooth surface is plotted in Figure (7.10). The forward movement step size for the real robot on a rough surface is closely resembled by the step size for the smooth surface. This similarity also applies for the simulated robot. However, there is a considerable discrepancy between the step size of the real robot compared to the simulated robot.

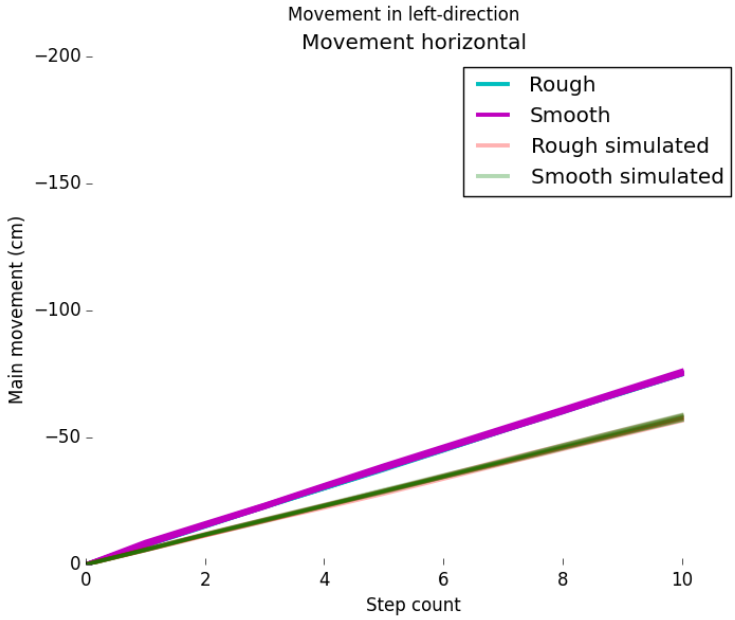

Figure 7.10: Movement forward on the rough surface, difference in x-direction plotted

In Figure (7.11) the vertical movement (error) is plotted of the left movement. As well as with the forward movement for the real robot, the smooth surface increases the variance for the error in the left movement (shown in purple compared to the cyan color). The simulated robot seems to have a decrease in variance.

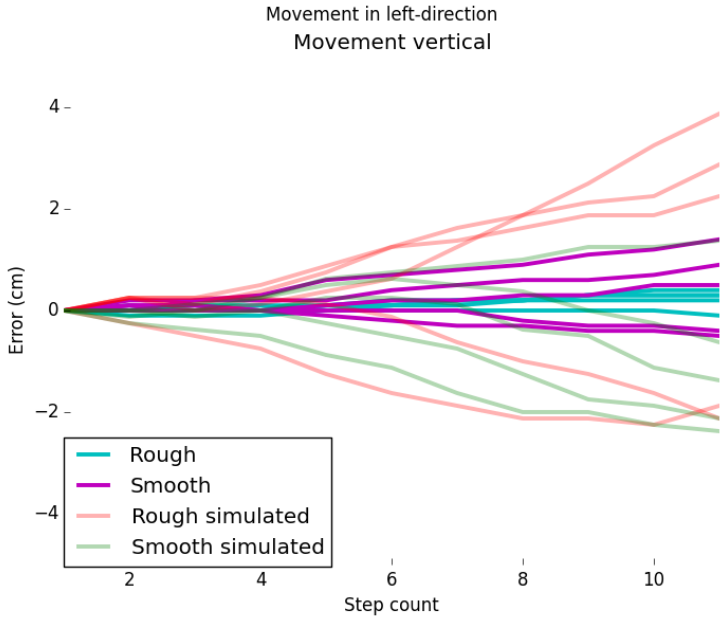

Figure 7.11: Movement forward on the rough surface, difference in y-direction plotted

7.6 Left movement for different elevations

The step size for the left movement on rough surface for 0, 1.5 and 4,5 degrees as ”Rough”, ”Slope 1” and ”Slope 2 ” is shown in Figure (7.12). For the real robot the step size is lower as elevation is higher. The difference between 0 and 1.5 is 1 cm and the difference between the angles 1.5

degrees and 4.5 degrees is ... cm after ten steps. This indicates that a 1.5 degrees in angle for the surface causes a linear decrease of 2 cm in step size. Also seen in the forward experiment, the step size of the simulated robot seems more effected by elevation. As the difference between 0 degrees and 1.5 degrees is 4 cm and the difference between 1.5 and 4.5 is 7 cm after ten steps. This is a linear decrease.

The left movement decrease is linear for both the real and simulated robot, in contrast to the exponential decrease of the forward movement on the elevated surfaces.

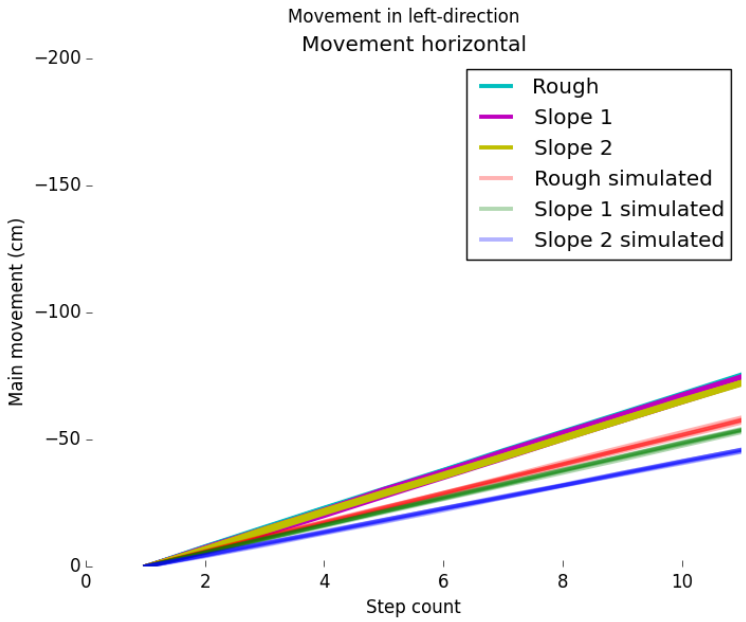

Figure 7.12: Movement left on three elevations, difference in y-direction plotted

In Figure (7.13) the error of the left movement on rough surfaces with elevations is plotted. Again the variance in the error of the simulated robot is higher. The real robot has an error distinctly in the forward-direction, whereas the simulated robot has errors in both the forward- and backward-direction.

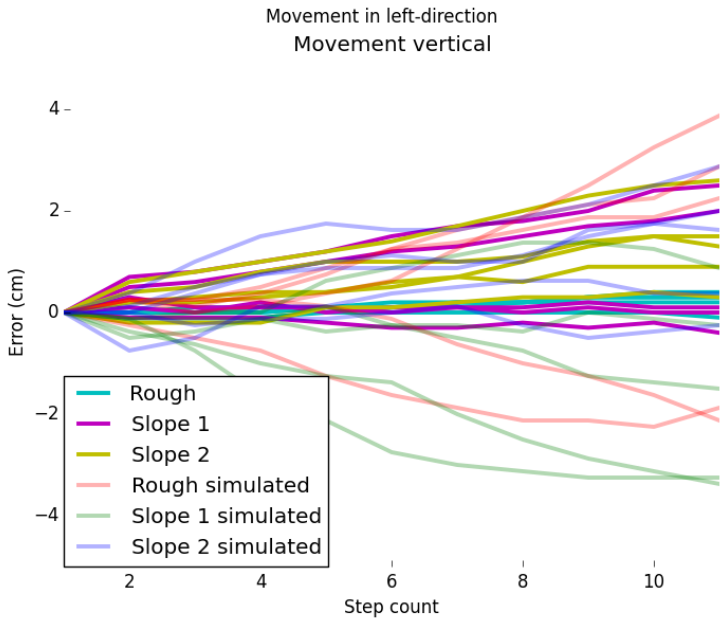

Figure 7.13: Movement left on three elevations, difference in x-direction plotted

CHAPTER 8

Discussion and future work

The real experiments were recorded by three people, it is possible that measurement were interpreted differently. As a precaution every experiment was measured by the same person so every different interpretation had the same relative difference. Also the surface was a carpet meaning that wrinkles and inconsistent bumps were possible. A countermeasure was that the carpet was stretched as much as possible and locked in place. Measurement were done by measuring tape, a better sensor would be a groundtruth sensor with a corresponding frequency of the USARSim sensor for results measured with a higher frequency.

The results show that the movement of the virtual KUKA youBot can be improved. Future research can be done by experimenting with a kinematic based implementation. This implementation has its movement based on the kinematic model of a platform with four Mecanum wheels. The movement of the vehicle will be provided by use of additional code and the movement of four wheels. The movement of the rollers of the Mecanum wheels are provided by the additional code. Kinematic based movement is also used in different simulators, e.g. in Gazebo, which will be discussed in related work, a KUKA youBot is implemented solely trough a kinematic model.

The use of the ROS wrapper to drive the real robot did complicate the interpretation of the results. The wrapper made the movement equal for a move forward and a move left. This is certainly not the case if wheel speeds were the same, the left movement should be less than the forward movement. Real results were scaled with the angle of the rollers but there still is a discrepancy. For further testing equivalent code should be used for both the real and simulated robot.

The results of the simulated KUKA youBot were often exponential, indicating a rotation in the robot. For further research rotation should be measured of the real robot to compare it to the rotation of the simulated one.

The weight of the simulated youBot did not seems to effect step size, but the center of gravity can seriously effect rotation as seen with previous results. The center of gravity of the real robot is not precisely in the center of the KUKA youBot base, this is result of the internal computer and motors. For further testing the center of gravity of the real robot can be measured and applied to the simulated robot.

Only the left, right forward and backward movement are tested. Mecanum wheels have a lot more possible moves, e.g. rotating in place or moving in a diagonal. A validation of all the moves is needed to provide a the movement of the omnidirectional drive based on Mecanum wheels.

A kinematic correction for the errors of the simulated robot can be applied. For the main movement, the simulation can simply scale its speeds. Also if a side-movement command is given, the speeds can be increased by

$$WheelSpeed_{simulation} = WheelSpeed_{simulation} \times \frac{2}{\sqrt{2}} \tag{8.1}$$

CHAPTER 9

Conclusion

In this thesis it is demonstrated that the simulation of the KUKA youBot in USARSim resembles the step size of the real one with a discrepancy. Further improvements should be made to narrow this discrepancy, however, the results seem positive since more parameters can be optimized. Optimizing of the physical attributes, center of gravity or even joint attributes are options. Also kinematic approach instead of the approximate implementation is worth undertaking as an alternative.

Future experiments could scale the main movement and reduce the errors of the simulated robot by parameter optimization. That would validate that USARSim supports the movement of an omnidirectional vehicle, serving as a research tool for a wide range of robots. Given the fact that the implementation only approximates the real robot and does not use any code, proves USARSim's potential. All in all the simulator based on the Unreal game engine provides a powerful combination with a focus on representing reality.

APPENDIX A

Robot code

```
class KUKA_youBot extends OmniSteeredVehicle config(USAR);


//wheel distances
`define WheelX .2355
`define WheelY .15
`define WheelZ -0.03

`define c1 0.0475
`define c2 0.02375
`define c3 0.04114
//
//
//roller distances
`define roller_1_x              `c1
`define roller_2_x              `c2
`define roller_3_x              -`c2
`define roller_4_x              -`c1
`define roller_5_x              -`c2
`define roller_6_x              `c2

`define roller_1_z              0
`define roller_2_z              `c3
`define roller_3_z              `c3
`define roller_4_z              0
`define roller_5_z              -`c3
`define roller_6_z              -`c3


//roller joint rotations, with incremental steps of -1/3 PI
`define roller_dir_y1   0
`define roller_dir_y2   -1.0471975512
`define roller_dir_y3   -2.09439510239
`define roller_dir_y4   -3.14159265359
`define roller_dir_y5   -4.18879020479
`define roller_dir_y6   -5.23598775598

`define pi_halve                1.57079632679
`define pi_third                1.0471975512
`define pi_fourth               0.78539816339


//create wheel meshes
`define Create_wheelMesh(MyObject, MyMesh, MyMass, MyX, MyY, MyZ) \
Begin Object Class=Part Name=`{MyObject}\n \
Mesh=`{MyMesh}\n \
Offset=(X=`{MyX},Y=`{MyY},Z=`{MyZ})\n \
Mass=`{MyMass}\n \
End Object\n \
PartList.Add(`{MyObject})\n \
```

```
//create roller meshes
`define Create_rollerMesh(MyObject, MyMesh, MyParent, MyMass, MyX, MyY, MyZ) \
Begin Object Class=Part Name=`{MyObject}\n \
Mesh=`{MyMesh}\n \
RelativeTo = `{MyParent}\n \
Offset=(X=`{MyX},Y=`{MyY},Z=`{MyZ})\n \
Mass=`{MyMass}\n \
End Object\n \
PartList.Add(`{MyObject})\n \

//create wheel joints
`define Create_wheelJoint(MyObject, MyChild, MySide, MyX, MyY, MyZ, MyDirX, MyDirY
    , MyDirZ) \
Begin Object Class=WheelJoint Name=`{MyObject}\n \
Parent=kukabody\n \
Child=`{MyChild}\n \
Side=`{MySide}\n \
Offset=(X=`{MyX},Y=`{MyY},Z=`{MyZ})\n \
Direction=(X=`{MyDirX},Y=`{MyDirY},Z=`{MyDirZ})\n \
MaxVelocity=2\n \
End Object\n \
Joints.Add(`{MyObject})\n \

//create connectionjoints
`define Create_connectionJoint(MyObject, MyChild, MyX, MyY, MyZ,) \
Begin Object Class=WheelJoint Name=`{MyObject}\n \
Parent=kukabody\n \
Child=`{MyChild}\n \
Offset=(X=`{MyX},Y=`{MyY},Z=`{MyZ})\n \
End Object\n \
Joints.Add(`{MyObject})\n \


//create roller joints
`define Create_rollerJoint(MyObject, MyParent, MyChild, MyX, MyY, MyZ, MyDirX,
    MyDirY, MyDirZ) \
Begin Object Class=RollerJoint Name=`{MyObject}\n \
Parent=`{MyParent}\n \
Child=`{MyChild}\n \
Side=SIDE_None\n \
RelativeTo =`{MyParent}\n \
Offset=(X=`{MyX},Y=`{MyY},Z=`{MyZ})\n \
Direction=(X=`{MyDirX},Y=`{MyDirY},Z=`{MyDirZ})\n \
MaxVelocity=1.3\n \
End Object\n \
Joints.Add(`{MyObject})\n \

`define Create_meshJoint(MyObject, MyParent, MyChild) \
Begin Object Class=RollerJoint Name=`{MyObject}\n \
Parent=`{MyParent}\n \
Child=`{MyChild}\n \
Offset=(X=0,Y=0,Z=0\n \
End Object\n \
Joints.Add(`{MyObject})\n \



defaultproperties
{


// Create body part
Begin Object Class=Part Name=kukabody
Mesh=StaticMesh'KUKA_youBot.youbot_base_frame'
Mass=8.6
End Object
Body=kukabody
PartList.Add(kukabody)
```

```
‘Create_wheelMesh(ArmJoint_1, StaticMesh'KUKA_youBot.youbot_arm_joint_1', 1, 0, 0,
    0)
‘Create_wheelMesh(ArmJoint_2, StaticMesh'KUKA_youBot.youbot_arm_joint_2', 1, 0, 0,
    0)
‘Create_wheelMesh(ArmJoint_3, StaticMesh'KUKA_youBot.youbot_arm_joint_3', 1, 0, 0,
    0)
‘Create_wheelMesh(ArmBase, StaticMesh'KUKA_youBot.youbot_arm_base_frame', 1, 0, 0,
    0)
‘Create_wheelMesh(Plate, StaticMesh'KUKA_youBot.youbot_plate', 1, 0, 0, 0)



/*
Create meshes for the wheels
FLWheel
FRWheel
BLWheel
BRWheel
*/
‘Create_wheelMesh(FLWheel, StaticMesh'KUKA_youBot.main_try', 1, ‘WheelX, -‘WheelY,
    ‘WheelZ)
‘Create_wheelMesh(FRWheel, StaticMesh'KUKA_youBot.main_try', 1, ‘WheelX, ‘WheelY,
    ‘WheelZ)
‘Create_wheelMesh(BLWheel, StaticMesh'KUKA_youBot.main_try', 1, -‘WheelX, -‘WheelY
    , ‘WheelZ)
‘Create_wheelMesh(BRWheel, StaticMesh'KUKA_youBot.main_try', 1, -‘WheelX, ‘WheelY,
    ‘WheelZ)



/*Create meshes for rollers
x,z needed for roller placement
MyObject, MyMesh, Myparent, MyMass, MyX, MyY, MyZ
*/
‘Create_rollerMesh(FLWheel_Roller_1, StaticMesh'KUKA_youBot.roller_3b', FLWheel,
    0.1, ‘roller_1_x, 0, ‘roller_1_z)
‘Create_rollerMesh(FLWheel_Roller_2, StaticMesh'KUKA_youBot.roller_3b2', FLWheel,
    0.1, ‘roller_2_x, 0, ‘roller_2_z)
‘Create_rollerMesh(FLWheel_Roller_3, StaticMesh'KUKA_youBot.roller_3b3', FLWheel,
    0.1, ‘roller_3_x, 0, ‘roller_3_z)
‘Create_rollerMesh(FLWheel_Roller_4, StaticMesh'KUKA_youBot.roller_3b4', FLWheel,
    0.1, ‘roller_4_x, 0, ‘roller_4_z)
‘Create_rollerMesh(FLWheel_Roller_5, StaticMesh'KUKA_youBot.roller_3b5', FLWheel,
    0.1, ‘roller_5_x, 0, ‘roller_5_z)
‘Create_rollerMesh(FLWheel_Roller_6, StaticMesh'KUKA_youBot.roller_3b6', FLWheel,
    0.1, ‘roller_6_x, 0, ‘roller_6_z)

‘Create_rollerMesh(FRWheel_Roller_1, StaticMesh'KUKA_youBot.roller_3a', FRWheel,
    0.1, ‘roller_1_x, 0, ‘roller_1_z)
‘Create_rollerMesh(FRWheel_Roller_2, StaticMesh'KUKA_youBot.roller_3a2', FRWheel,
    0.1, ‘roller_2_x, 0, ‘roller_2_z)
‘Create_rollerMesh(FRWheel_Roller_3, StaticMesh'KUKA_youBot.roller_3a3', FRWheel,
    0.1, ‘roller_3_x, 0, ‘roller_3_z)
‘Create_rollerMesh(FRWheel_Roller_4, StaticMesh'KUKA_youBot.roller_3a4', FRWheel,
    0.1, ‘roller_4_x, 0, ‘roller_4_z)
‘Create_rollerMesh(FRWheel_Roller_5, StaticMesh'KUKA_youBot.roller_3a5', FRWheel,
    0.1, ‘roller_5_x, 0, ‘roller_5_z)
‘Create_rollerMesh(FRWheel_Roller_6, StaticMesh'KUKA_youBot.roller_3a6', FRWheel,
    0.1, ‘roller_6_x, 0, ‘roller_6_z)

‘Create_rollerMesh(BLWheel_Roller_1, StaticMesh'KUKA_youBot.roller_3a', BLWheel,
    0.1, ‘roller_1_x, 0, ‘roller_1_z)
‘Create_rollerMesh(BLWheel_Roller_2, StaticMesh'KUKA_youBot.roller_3a2', BLWheel,
    0.1, ‘roller_2_x, 0, ‘roller_2_z)
‘Create_rollerMesh(BLWheel_Roller_3, StaticMesh'KUKA_youBot.roller_3a3', BLWheel,
    0.1, ‘roller_3_x, 0, ‘roller_3_z)
‘Create_rollerMesh(BLWheel_Roller_4, StaticMesh'KUKA_youBot.roller_3a4', BLWheel,
    0.1, ‘roller_4_x, 0, ‘roller_4_z)
‘Create_rollerMesh(BLWheel_Roller_5, StaticMesh'KUKA_youBot.roller_3a5', BLWheel,
    0.1, ‘roller_5_x, 0, ‘roller_5_z)
```

```
`Create_rollerMesh(BLWheel_Roller_6, StaticMesh'KUKA_youBot.roller_3a6', BLWheel,
    0.1, `roller_6_x, 0, `roller_6_z)

`Create_rollerMesh(BRWheel_Roller_1, StaticMesh'KUKA_youBot.roller_3b', BRWheel,
    0.1, `roller_1_x, 0, `roller_1_z)
`Create_rollerMesh(BRWheel_Roller_2, StaticMesh'KUKA_youBot.roller_3b2', BRWheel,
    0.1, `roller_2_x, 0, `roller_2_z)
`Create_rollerMesh(BRWheel_Roller_3, StaticMesh'KUKA_youBot.roller_3b3', BRWheel,
    0.1, `roller_3_x, 0, `roller_3_z)
`Create_rollerMesh(BRWheel_Roller_4, StaticMesh'KUKA_youBot.roller_3b4', BRWheel,
    0.1, `roller_4_x, 0, `roller_4_z)
`Create_rollerMesh(BRWheel_Roller_5, StaticMesh'KUKA_youBot.roller_3b5', BRWheel,
    0.1, `roller_5_x, 0, `roller_5_z)
`Create_rollerMesh(BRWheel_Roller_6, StaticMesh'KUKA_youBot.roller_3b6', BRWheel,
    0.1, `roller_6_x, 0, `roller_6_z)



//Self collision is turned off
//Collision within the main object
DisableContacts.Add((Part1=ArmBase,Part2=kukabody))
DisableContacts.Add((Part1=Plate,Part2=kukabody))
DisableContacts.Add((Part1=ArmBase,Part2=ArmJoint_1))
DisableContacts.Add((Part1=ArmJoint_1,Part2=ArmJoint_2))

DisableContacts.Add((Part1=ArmJoint_2,Part2=ArmJoint_3))

//Collision between wheels and body
DisableContacts.Add((Part1=FLWheel,Part2=kukabody))
DisableContacts.Add((Part1=FRWheel,Part2=kukabody))
DisableContacts.Add((Part1=BLWheel,Part2=kukabody))
DisableContacts.Add((Part1=BRWheel,Part2=kukabody))

//Collision between wheels and rollers
DisableContacts.Add((Part1=FLWheel,Part2=FLWheel_Roller_1))
DisableContacts.Add((Part1=FLWheel,Part2=FLWheel_Roller_2))
DisableContacts.Add((Part1=FLWheel,Part2=FLWheel_Roller_3))
DisableContacts.Add((Part1=FLWheel,Part2=FLWheel_Roller_4))
DisableContacts.Add((Part1=FLWheel,Part2=FLWheel_Roller_5))
DisableContacts.Add((Part1=FLWheel,Part2=FLWheel_Roller_6))

DisableContacts.Add((Part1=FRWheel,Part2=FRWheel_Roller_1))
DisableContacts.Add((Part1=FRWheel,Part2=FRWheel_Roller_2))
DisableContacts.Add((Part1=FRWheel,Part2=FRWheel_Roller_3))
DisableContacts.Add((Part1=FRWheel,Part2=FRWheel_Roller_4))
DisableContacts.Add((Part1=FRWheel,Part2=FRWheel_Roller_5))
DisableContacts.Add((Part1=FRWheel,Part2=FRWheel_Roller_6))

DisableContacts.Add((Part1=BLWheel,Part2=BLWheel_Roller_1))
DisableContacts.Add((Part1=BLWheel,Part2=BLWheel_Roller_2))
DisableContacts.Add((Part1=BLWheel,Part2=BLWheel_Roller_3))
DisableContacts.Add((Part1=BLWheel,Part2=BLWheel_Roller_4))
DisableContacts.Add((Part1=BLWheel,Part2=BLWheel_Roller_5))
DisableContacts.Add((Part1=BLWheel,Part2=BLWheel_Roller_6))

DisableContacts.Add((Part1=BRWheel,Part2=BRWheel_Roller_1))
DisableContacts.Add((Part1=BRWheel,Part2=BRWheel_Roller_2))
DisableContacts.Add((Part1=BRWheel,Part2=BRWheel_Roller_3))
DisableContacts.Add((Part1=BRWheel,Part2=BRWheel_Roller_4))
DisableContacts.Add((Part1=BRWheel,Part2=BRWheel_Roller_5))
DisableContacts.Add((Part1=BRWheel,Part2=BRWheel_Roller_6))


//Collision between body and rollers
DisableContacts.Add((Part1=kukabody,Part2=FLWheel_Roller_1))
DisableContacts.Add((Part1=kukabody,Part2=FLWheel_Roller_2))
DisableContacts.Add((Part1=kukabody,Part2=FLWheel_Roller_3))
DisableContacts.Add((Part1=kukabody,Part2=FLWheel_Roller_4))
DisableContacts.Add((Part1=kukabody,Part2=FLWheel_Roller_5))
DisableContacts.Add((Part1=kukabody,Part2=FLWheel_Roller_6))
```

```
DisableContacts.Add((Part1=kukabody,Part2=FRWheel_Roller_1))
DisableContacts.Add((Part1=kukabody,Part2=FRWheel_Roller_2))
DisableContacts.Add((Part1=kukabody,Part2=FRWheel_Roller_3))
DisableContacts.Add((Part1=kukabody,Part2=FRWheel_Roller_4))
DisableContacts.Add((Part1=kukabody,Part2=FRWheel_Roller_5))
DisableContacts.Add((Part1=kukabody,Part2=FRWheel_Roller_6))

DisableContacts.Add((Part1=kukabody,Part2=BLWheel_Roller_1))
DisableContacts.Add((Part1=kukabody,Part2=BLWheel_Roller_2))
DisableContacts.Add((Part1=kukabody,Part2=BLWheel_Roller_3))
DisableContacts.Add((Part1=kukabody,Part2=BLWheel_Roller_4))
DisableContacts.Add((Part1=kukabody,Part2=BLWheel_Roller_5))
DisableContacts.Add((Part1=kukabody,Part2=BLWheel_Roller_6))

DisableContacts.Add((Part1=kukabody,Part2=BRWheel_Roller_1))
DisableContacts.Add((Part1=kukabody,Part2=BRWheel_Roller_2))
DisableContacts.Add((Part1=kukabody,Part2=BRWheel_Roller_3))
DisableContacts.Add((Part1=kukabody,Part2=BRWheel_Roller_4))
DisableContacts.Add((Part1=kukabody,Part2=BRWheel_Roller_5))
DisableContacts.Add((Part1=kukabody,Part2=BRWheel_Roller_6))

//Collision between rollers
DisableContacts.Add((Part1=FLWheel_Roller_6,Part2=FLWheel_Roller_1))
DisableContacts.Add((Part1=FLWheel_Roller_1,Part2=FLWheel_Roller_2))
DisableContacts.Add((Part1=FLWheel_Roller_2,Part2=FLWheel_Roller_3))
DisableContacts.Add((Part1=FLWheel_Roller_3,Part2=FLWheel_Roller_4))
DisableContacts.Add((Part1=FLWheel_Roller_4,Part2=FLWheel_Roller_5))
DisableContacts.Add((Part1=FLWheel_Roller_5,Part2=FLWheel_Roller_6))

DisableContacts.Add((Part1=FRWheel_Roller_6,Part2=FRWheel_Roller_1))
DisableContacts.Add((Part1=FRWheel_Roller_1,Part2=FRWheel_Roller_2))
DisableContacts.Add((Part1=FRWheel_Roller_2,Part2=FRWheel_Roller_3))
DisableContacts.Add((Part1=FRWheel_Roller_3,Part2=FRWheel_Roller_4))
DisableContacts.Add((Part1=FRWheel_Roller_4,Part2=FRWheel_Roller_5))
DisableContacts.Add((Part1=FRWheel_Roller_5,Part2=FRWheel_Roller_6))

DisableContacts.Add((Part1=BLWheel_Roller_6,Part2=BLWheel_Roller_1))
DisableContacts.Add((Part1=BLWheel_Roller_1,Part2=BLWheel_Roller_2))
DisableContacts.Add((Part1=BLWheel_Roller_2,Part2=BLWheel_Roller_3))
DisableContacts.Add((Part1=BLWheel_Roller_3,Part2=BLWheel_Roller_4))
DisableContacts.Add((Part1=BLWheel_Roller_4,Part2=BLWheel_Roller_5))
DisableContacts.Add((Part1=BLWheel_Roller_5,Part2=BLWheel_Roller_6))

DisableContacts.Add((Part1=BRWheel_Roller_6,Part2=BRWheel_Roller_1))
DisableContacts.Add((Part1=BRWheel_Roller_1,Part2=BRWheel_Roller_2))
DisableContacts.Add((Part1=BRWheel_Roller_2,Part2=BRWheel_Roller_3))
DisableContacts.Add((Part1=BRWheel_Roller_3,Part2=BRWheel_Roller_4))
DisableContacts.Add((Part1=BRWheel_Roller_4,Part2=BRWheel_Roller_5))
DisableContacts.Add((Part1=BRWheel_Roller_5,Part2=BRWheel_Roller_6))


//Create wheel joints
`Create_wheelJoint(FLWheelRoll, FLWheel, SIDE_Front_Left, `WheelX, -`WheelY, `
    WheelZ, `pi_halve, 0, 0)
`Create_wheelJoint(FRWheelRoll, FRWheel, SIDE_Front_Right, `WheelX, `WheelY, `
    WheelZ, `pi_halve, 0, 0)
`Create_wheelJoint(BLWheelRoll, BLWheel, SIDE_Back_Left, -`WheelX, -`WheelY, `
    WheelZ, `pi_halve, 0, 0)
`Create_wheelJoint(BRWheelRoll, BRWheel, SIDE_Back_Right, -`WheelX, `WheelY, `
    WheelZ, `pi_halve, 0, 0)

`Create_connectionJoint(Conn, ArmJoint_1, 0, 0, 0)
`Create_connectionJoint(Conn2, ArmJoint_2, 0, 0, 0)
`Create_connectionJoint(Conn3, ArmJoint_3, 0, 0, 0)
`Create_connectionJoint(Conn6, ArmBase, 0, 0, 0)
`Create_connectionJoint(Conn10, Plate, 0, 0, 0)

//Create roller joints
`Create_rollerJoint(FLWheelRoller_001, FLWheel, FLWheel_Roller_1, `roller_1_x, 0,
    `roller_1_z, `pi_fourth, `roller_dir_y1, 0)
`Create_rollerJoint(FLWheelRoller_002, FLWheel, FLWheel_Roller_2, `roller_2_x, 0,
```

```
    `roller_2_z , `pi_fourth , `roller_dir_y2 , 0)
`Create_rollerJoint(FLWheelRoller_003, FLWheel, FLWheel_Roller_3, `roller_3_x , 0,
    `roller_3_z , `pi_fourth , `roller_dir_y3 , 0)
`Create_rollerJoint(FLWheelRoller_004, FLWheel, FLWheel_Roller_4, `roller_4_x , 0,
    `roller_4_z , `pi_fourth , `roller_dir_y4 , 0)
`Create_rollerJoint(FLWheelRoller_005, FLWheel, FLWheel_Roller_5, `roller_5_x , 0,
    `roller_5_z , `pi_fourth , `roller_dir_y5 , 0)
`Create_rollerJoint(FLWheelRoller_006, FLWheel, FLWheel_Roller_6, `roller_6_x , 0,
    `roller_6_z , `pi_fourth , `roller_dir_y6 , 0)

`Create_rollerJoint(FRWheelRoller_001, FRWheel, FRWheel_Roller_1, `roller_1_x , 0,
    `roller_1_z , -`pi_fourth , `roller_dir_y1 , 0)
`Create_rollerJoint(FRWheelRoller_002, FRWheel, FRWheel_Roller_2, `roller_2_x , 0,
    `roller_2_z , -`pi_fourth , `roller_dir_y2 , 0)
`Create_rollerJoint(FRWheelRoller_003, FRWheel, FRWheel_Roller_3, `roller_3_x , 0,
    `roller_3_z , -`pi_fourth , `roller_dir_y3 , 0)
`Create_rollerJoint(FRWheelRoller_004, FRWheel, FRWheel_Roller_4, `roller_4_x , 0,
    `roller_4_z , -`pi_fourth , `roller_dir_y4 , 0)
`Create_rollerJoint(FRWheelRoller_005, FRWheel, FRWheel_Roller_5, `roller_5_x , 0,
    `roller_5_z , -`pi_fourth , `roller_dir_y5 , 0)
`Create_rollerJoint(FRWheelRoller_006, FRWheel, FRWheel_Roller_6, `roller_6_x , 0,
    `roller_6_z , -`pi_fourth , `roller_dir_y6 , 0)

`Create_rollerJoint(BLWheelRoller_001, BLWheel, BLWheel_Roller_1, `roller_1_x , 0,
    `roller_1_z , -`pi_fourth , `roller_dir_y1 , 0)
`Create_rollerJoint(BLWheelRoller_002, BLWheel, BLWheel_Roller_2, `roller_2_x , 0,
    `roller_2_z , -`pi_fourth , `roller_dir_y2 , 0)
`Create_rollerJoint(BLWheelRoller_003, BLWheel, BLWheel_Roller_3, `roller_3_x , 0,
    `roller_3_z , -`pi_fourth , `roller_dir_y3 , 0)
`Create_rollerJoint(BLWheelRoller_004, BLWheel, BLWheel_Roller_4, `roller_4_x , 0,
    `roller_4_z , -`pi_fourth , `roller_dir_y4 , 0)
`Create_rollerJoint(BLWheelRoller_005, BLWheel, BLWheel_Roller_5, `roller_5_x , 0,
    `roller_5_z , -`pi_fourth , `roller_dir_y5 , 0)
`Create_rollerJoint(BLWheelRoller_006, BLWheel, BLWheel_Roller_6, `roller_6_x , 0,
    `roller_6_z , -`pi_fourth , `roller_dir_y6 , 0)

`Create_rollerJoint(BRWheelRoller_001, BRWheel, BRWheel_Roller_1, `roller_1_x , 0,
    `roller_1_z , `pi_fourth , `roller_dir_y1 , 0)
`Create_rollerJoint(BRWheelRoller_002, BRWheel, BRWheel_Roller_2, `roller_2_x , 0,
    `roller_2_z , `pi_fourth , `roller_dir_y2 , 0)
`Create_rollerJoint(BRWheelRoller_003, BRWheel, BRWheel_Roller_3, `roller_3_x , 0,
    `roller_3_z , `pi_fourth , `roller_dir_y3 , 0)
`Create_rollerJoint(BRWheelRoller_004, BRWheel, BRWheel_Roller_4, `roller_4_x , 0,
    `roller_4_z , `pi_fourth , `roller_dir_y4 , 0)
`Create_rollerJoint(BRWheelRoller_005, BRWheel, BRWheel_Roller_5, `roller_5_x , 0,
    `roller_5_z , `pi_fourth , `roller_dir_y5 , 0)
`Create_rollerJoint(BRWheelRoller_006, BRWheel, BRWheel_Roller_6, `roller_6_x , 0,
    `roller_6_z , `pi_fourth , `roller_dir_y6 , 0)


WheelRadius=0.09
}
```

APPENDIX B

Parameter optimization

The friction value chosen for the experiments was 0.90. These were the compared frictions. Friction 0.70 seemed pretty good but had one out-lier. Friction 1.00 had too much variance. Friction 0.80 had one more error in the wrong direction than friction 0.90. All in all friction 0.90 was chosen with just one sample in the wrong error direction, which was at a late time step.

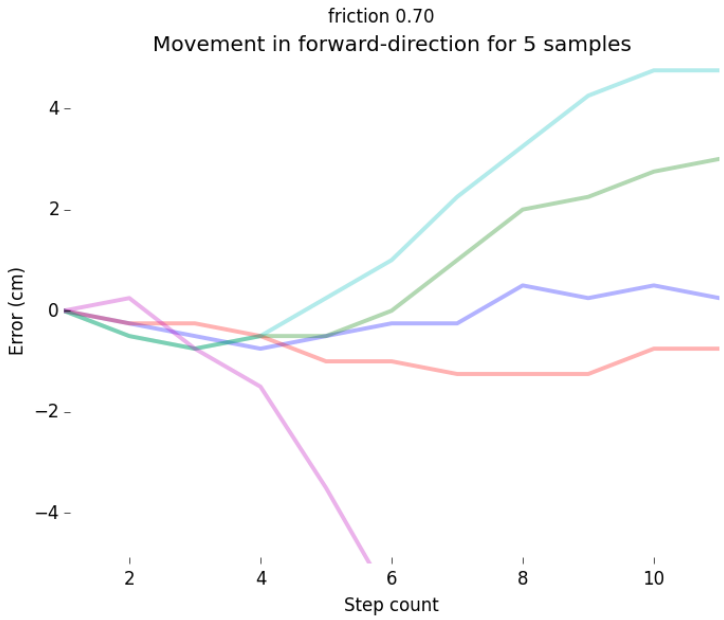

Figure B.1: Forward movement on different surfaces

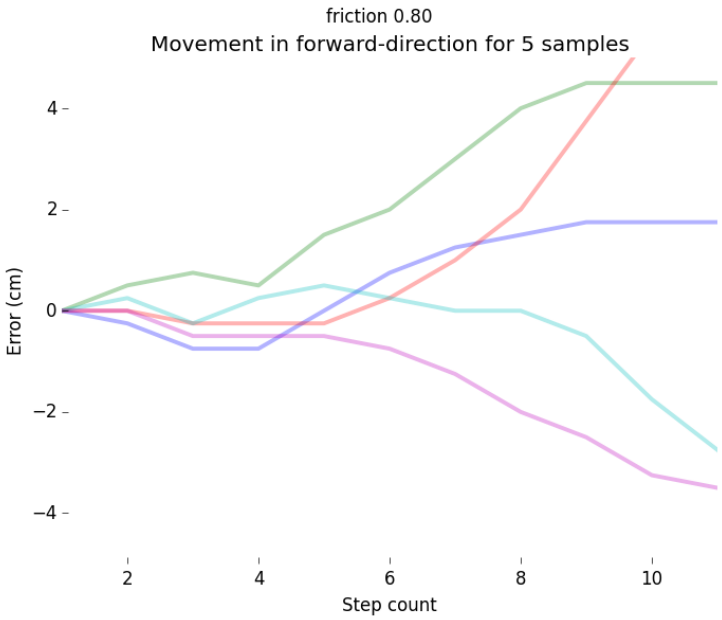

Figure B.2: Forward movement on different surfaces

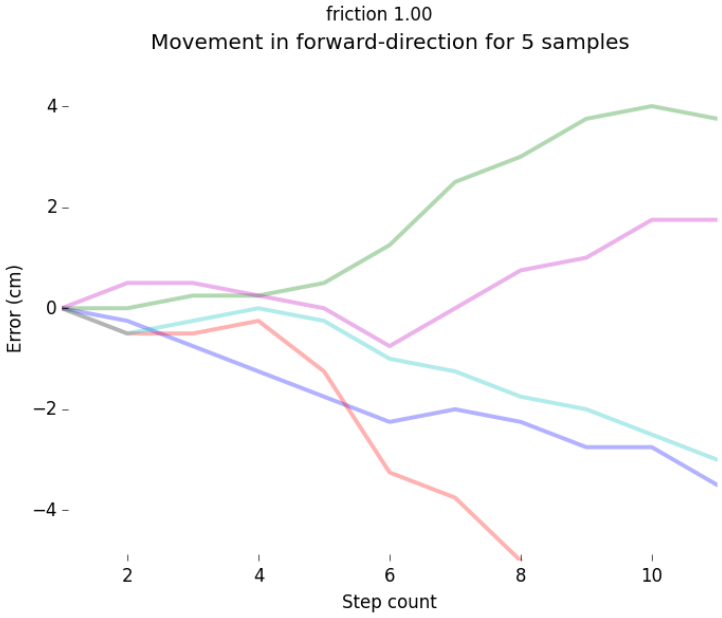

Figure B.3: Forward movement on different surfaces

APPENDIX C

Result with a incorrect center of gravity

These are the old simulation results compared to the new one. The old results had an incorrect center of gravity because part of the weight of the chassis was assigned to the robot arm. In the Figures below the old results are shown with a "with_arm" label. The main movement did slightly suffer because of the incorrect center of gravity seen in Figures (C.1), (C.3), (C.5) and (C.7). However, the error movement was exponential for all side movement, seen in Figures (C.6) and (C.8)

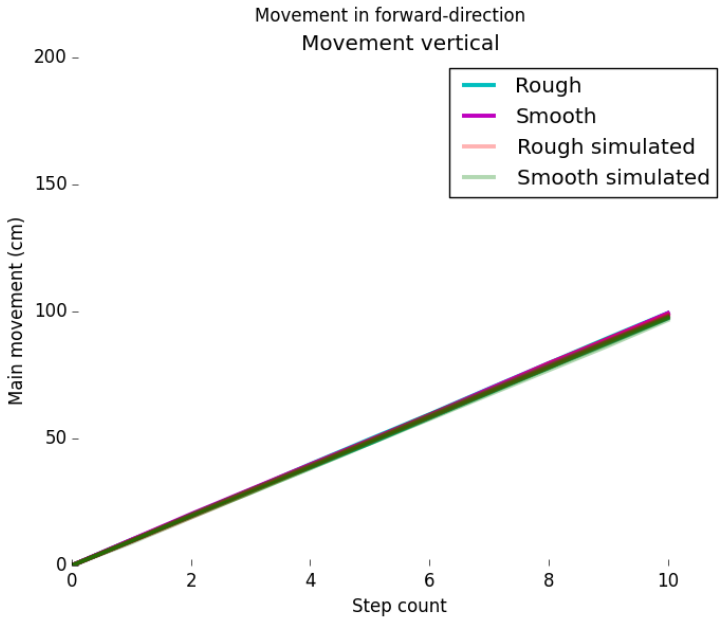

Figure C.1: Vertical movement for forward experiment on different surfaces

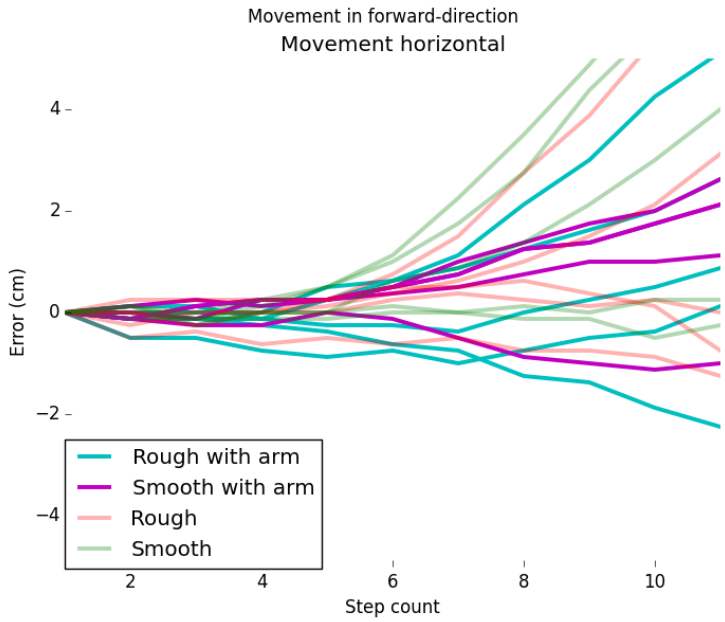

Figure C.2: Horizontal movement for forward experiment on different surfaces

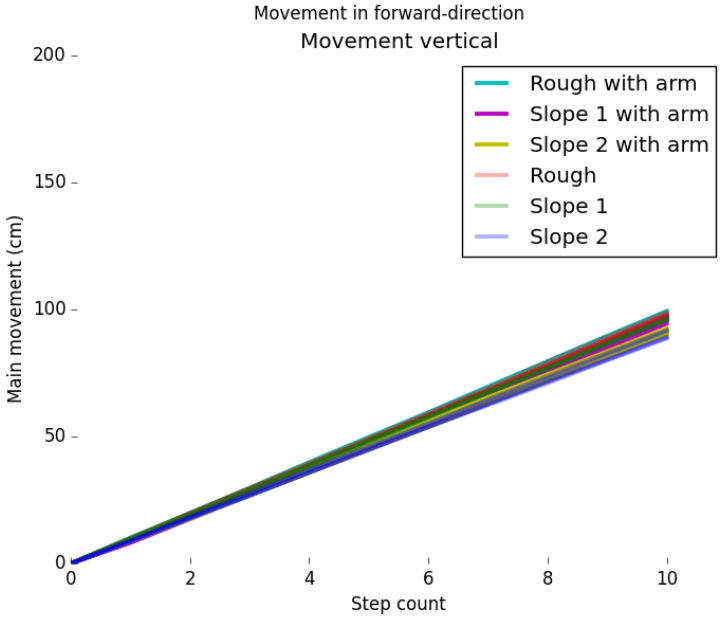

Figure C.3: Vertical movement for forward experiment on different elevations

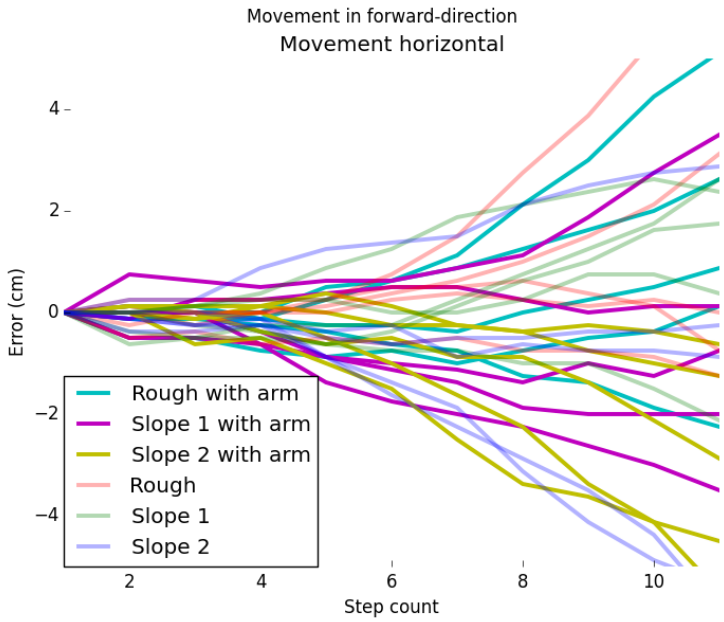

Figure C.4: Horizontal movement for forward experiment on different elevations

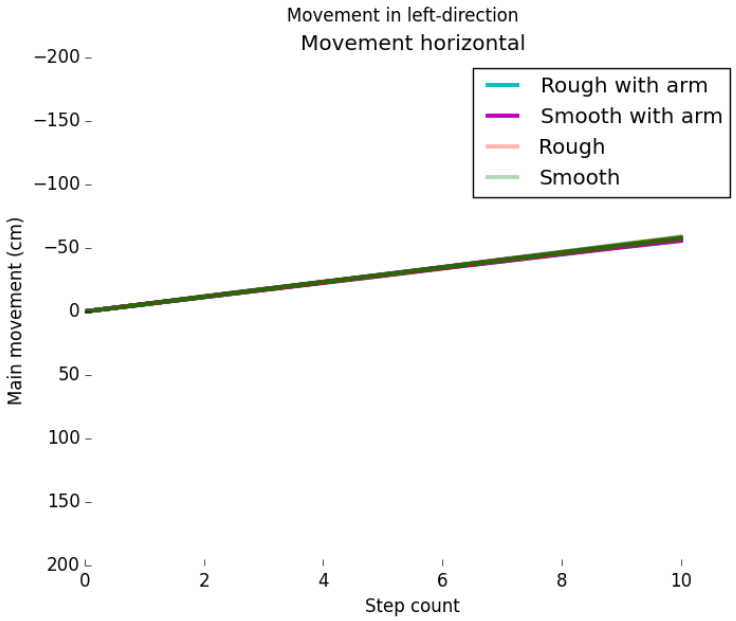

Figure C.5: Horizontal movement for left experiment on different surfaces

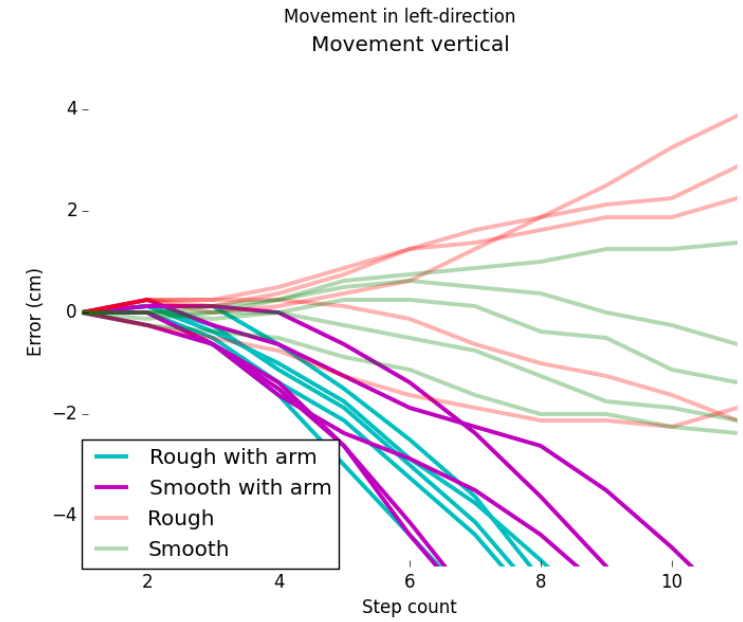

Figure C.6: Vertical movement for left experiment on different surfaces

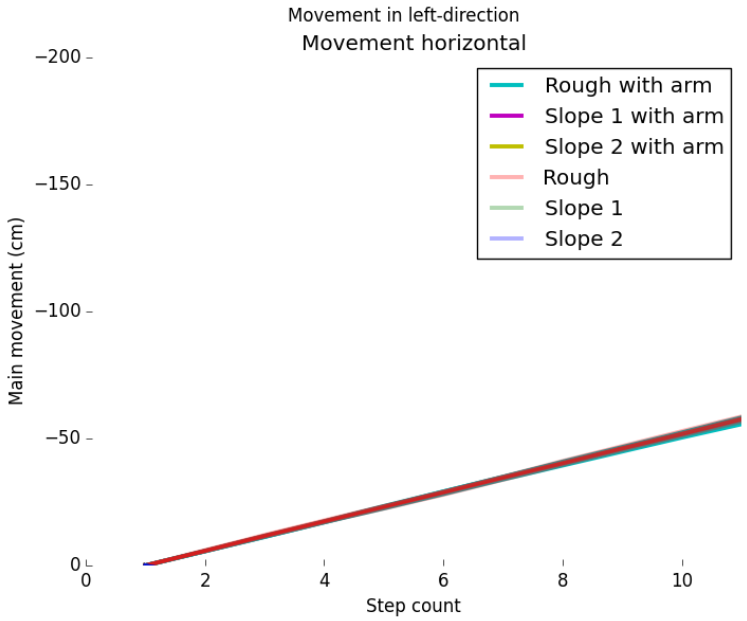

Figure C.7: Horizontal movement for left experiment on different elevations

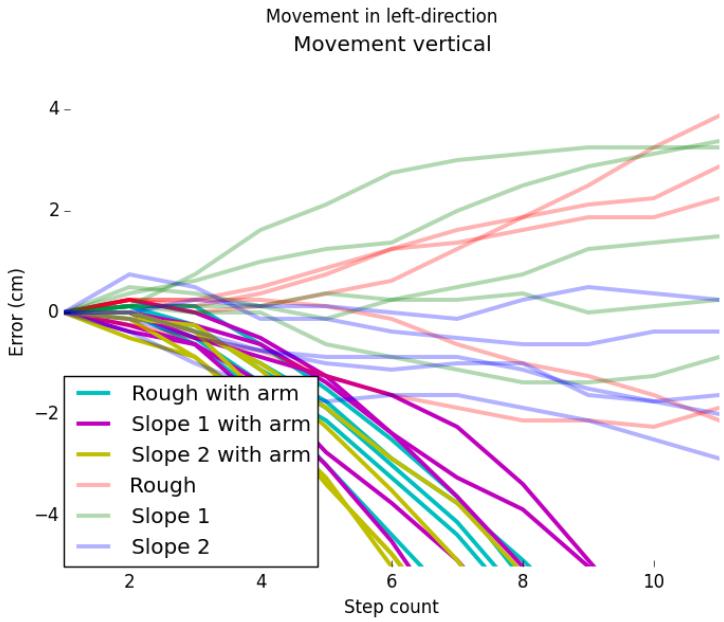

Figure C.8: Vertical movement for left experiment on different elevations

APPENDIX D

Notions

D.1 Open Source

An open source project is a project, which has the development model of universal access via a free license to a product's design or blueprint, and universal redistribution of that design or blueprint, including subsequent improvements to it by anyone. An open source project is an ongoing evolution of the project's first distribution. When a project is open source, contributions to the source code can come from the community supporting the project. When a project is proprietary only the developers can contribute to the project's source code. In simulators, being open source is beneficial for the development of the simulator since contributions can come from the community.

References

[1] P.F. Muir and C.P. Neuman. Wheel Types. In *Kinematic Modeling of Wheeled Mobile Robots, Camegie-Mellon University, Pittsburgh, Pennsylvania, 1996*

[2] *Webots User Guide* Cyberbotics Ltd, May 12, 2015.

[3] P.F. Muir and C.P. Neuman. *Kinematic modeling for feedback control of an omnidirectional wheeled mobile robot* Journal of robotic systems 4.2, Camegie-Mellon University, Pittsburgh, 1987.

[4] *KUKA youBot User Manual* December 6, 2012.

[5] Nathan K. and A. Howard. *Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator* Robotics Research Labs, University of Southern California, Los Angeles, USA.

[6] T. Laue and Thomas Röfer. *SimRobot - Development and Applications* Deutsches Forschungszentrum für Knstliche Intelligenz GmbH, Sichere Kognitive Systeme, Bremen, Germany.

[7] T. Laue, K. Spiess, and T. Röfer. *SimRobot - A General Physical Robot Simulator and its Application in RoboCup* Bremer Institut für Sichere Systeme, Technologie-Zentrum Informatik, FB 3, Universität Bremen, Bremen, Germany.

[8] D. Dresscher, Y. Brodskiy, P. Breedveld, J. Broenink and S. Stramigioli. *Modeling of the youBot in a serial link structure using twists and wrenches in a bond graph* University of Twente

[9] S. van Noort and A. Visser. *Validation of the dynamics of an humanoid robot in USARSim* Intelligent System Lab Amsterdam.

[10] S. Carpin, J. Wang, M. Lewis, A. Birk and A. Jacoff. *High fidelity tools for rescue robotics: results and perspectives* School of Engineering and Science, International University Bremen, Department of Information Science and Telecommunications, University of Pittsburgh, Intelligent Systems Division, National Institute of Standards and Technology

[11] G. Echeverria, N. Lassabe, A. Degroote and S. Lemaignan. *Modular Open Robots Simulation Engine: MORSE* Robotics and Automation (ICRA), IEEE International Conference on. IEEE, 2011.

[12] S. Carpin, T. Stoyanov, Y. Nevatia, M. Lewis and J. Wang. *Quantitative assessments of USARSim accuracy* International University Bremen, University of Pittsburgh, Proceedings of PerMIS. Vol. 2006, 2006.

[13] R. Siegwart, I. Nourbakhsh and Davide Scaramuzza. *Introduction to autonomous mobile robots* MIT press, 2011.

[14] A. Visser, N. Dijkshoorn, M. van der Veen and R. Jurriaans. *Closing the gap between simulation and reality in the sensor and motion models of an autonomous AR.Drone* International Micro Air Vehicle conference and competitions 2011 (IMAV 2011), Delft University of Technology and Thales, 2011.

[15] B. Balaguer, S. Balkirsky, S. Carpin, M. Lewis and C. Scrapper. *USARSim: a validated simulator for research in robotics and automation* University of California, University of Pittsburgh, National Institute of Standards and Technology, Workshop on Robot Simulators: Available Software, Scientific Applications, and Future Trends at IEEE/RSJ, 2008.

[16] S. Okamoto, K. Kurose, S. Saga, K. Ohno and S. Tadokoro. *Validation of Simulated Robots with Realistically Modeled Dimensions and Mass in USARSim* Tohoku University, Sendai, Japan.

[17] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper. *USARSim: a robot simulator for research and education* University of California, University of Pittsburgh, National Institute of Standards and Technology

[18] M. Lewis, S. Carpin and S. Balakirsky. *Virtual Robots RoboCupRescue Competition*: *Contributions to Infrastructure and Science*

[19] J. Busky, Z. Parrish and J. Wilson. Introduction to Unreal Technology. In *Mastering Unreal Technology Volume 1: Introduction to Level Design with Unreal Engine 3*

[20] R. Vaughan. *Massively Multi-robot simulation in stage*. Swarm Intell, 2:189-208, 1994.

[21] M. Lewis and J. Jacobson. *Game Engines In Scientific Research*. Communications of the ACM, 2002.

[22] M.W. Spong, S. Hutchinson and M. Vidyasagar *Robot Modeling and Control*. 2006.

[23] B. Sicilia and O. Khatib. *Kinematics* in *Handbook of Robotics*. Universita degli Studi di Napoli Federico II, Standford University, 2008.

[24] K. Morgan. *Introduction to Structured Programming using Turbo Pascal Version 5.0 on the IBM PC* Merrill Publishing Company, 1989.

[25] J. Wang and S. Balakirsky. *USARSim V3.1.3: A Game-based Simulation of mobile robots*

[26] R. Bischoff, U. Huggenberger and E. Prassler. *Kuka youbot-a mobile manipulator for research and education*. Robotics and Automation (ICRA), 2011 IEEE International Conference, 2011.

[27] O. Diegel, A. Badve, G. Bright, J. Potgieter and S. Tlake. *Improved Mecanum Wheel Design for Omni-directional Robots* Mechatronics and Robotics Research Group Institute of Technology and Engineering Massey University, Auckland.

[28] K. Capek. *Rossum's Universal Robots* 1921.

[29] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper. *USARSim: a robot simulator for research and education* .

[30] N. Dijkshoorn and A. Visser. *Integrating sensor and motion models to localize an autonomous AR.Drone* FNWI: Informatics Institute (II), University of Amsterdam, 2011.