LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
LUT School of Energy Systems
LUT Mechanical Engineering

Andres Belzunce

DEVELOPMENT OF A CONTROL SYSTEM FOR A TELE OPERATED MOBILE ROBOT

Examiners: Professor Heikki Handroos
Ph.D. Ming Li

ABSTRACT

Lappeenranta University of Technology
LUT School of Energy Systems
LUT Mechanical Engineering

Andres Belzunce

Development of a Control System for a Tele Operated Mobile Robot

Master's thesis

2015

56 pages, 36 figures, 2 tables and 2 appendices

Examiners: Professor Heikki Handroos
Ph.D. Ming Li
Keywords: ROS, Control System, Mobile Robot, Tip-Over Stability

Mobile robots are capable of performing spatial displacement motions in different environments. This motions can be calculated based on sensorial data (autonomous robot) or given by an operator (tele operated robot). This thesis is focused on the latter providing the control architecture which bridges the tele operator and the robot's locomotion system and end effectors. Such a task might prove overwhelming in cases where the robot comprises a wide variety of sensors and actuators hence a relatively new option was selected: Robot Operating System (ROS). The control system of a new robot will be sketched and tested in a simulation model using ROS together with Gazebo in order to determine the viability of such a system. The simulated model will be based on the projected shape and main features of the real machine. A stability analysis will be performed first theoretically and afterwards using the developed model.

This thesis concluded that both the physical properties and the control architecture are feasible and stable settling up the ground for further work with the same robot.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Ming Li for his advises and useful comments as well as Professor Heikki Handroos for granting me the freedom, time and opportunities required to learn ROS. Furthermore I would like to thank the other members of the mobile robot project without which the project would not have reached the state in which it is now.

Andres Belzunce
Lappeenranta 11.23.2015

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

$\hat{a}_i$	Contact point i axis
f_r	Resultant force [N]
f_i^*	Combination of resultant force and force couple equivalent [N]
I_i	Normal vector to $\hat{a}_i$
n_r	Resultant torque [N·m]
p_i	Contact point between system and ground
v_x	Robot's base velocity in Cartesian x direction [m/s]
v_y	Robot's base velocity in Cartesian y direction [m/s]
α_i	Tip-over margin around i-axis [N]
$\dot{\phi}_i$	Rotational speed of wheel i [Rad/s]
θ_i	Tip-over angle around i-axis [Rad]
ω_z	Rotational speed along Cartesian z axis [Rad/s]
AI	Artificial Intelligence
CoM	Center of Mass
EC	Electronically Commutated
GUI	Graphical User Interface
IDE	Integrated Development Environment
IMU	Inertial Navigation Unit
OS	Operating System
OROCOS	Open Robot Control Software
ROS	Robot Operating System
URDF	Unified Robot Description Format
VI	Visual Interface

1 INTRODUCTION

This section will introduce the system to be working on and the problems to be overcome. The robot to be built and studied will be described further giving some insight on the tasks to be performed. The main focus of the thesis will be outlined as well as the obtained scientific knowledge after its completion.

1.1 Background

Environments without continuous human supervision like automated manufacturing lines are becoming less and less human-friendly as technology advances. Chemical plants with noxious gases, high doses of radiation or highly exothermic chemical reactions follow the same category. With these environments in mind how can the human factor be introduced into the mix? The answer is simple: Telepresence.

As the name suggests the concept of telepresence is feeling or projecting the sensation of being in a place while actually being somewhere else. For the previously introduced problem it would be equivalent to having a human somewhere safe while the necessary sensorial information for that individual to feel the remote environment was fed to him through a computer. If the human were not only to feel being there but also were to act on the remote environment then a surrogate of some sort (actuator) would need to receive such commands. That is where the robot subject of this thesis comes in: it will receive sensorial information through cameras, microphones and distance sensors feeding them to the remote operator and actuate through two robot arms and hands based on the operator's requests.

The concept can be explained as a simple control engineering closed loop as shown in Figure 1.1 the reference signal would be the operator's desired action which would then be compared to what can be perceived through the sensor's signals. If they were to be different the operator would send a new command back to the robot so that the actuators will perform the desired action. At the same time the sensorial information would be fed back to the operator which will again decide what to do.

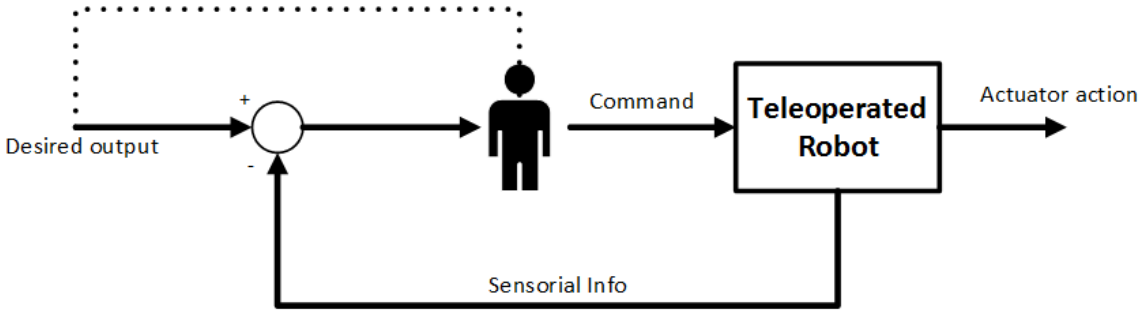

Figure 1.1. Telepresence concept.

The robot used for this research was initially designed for tele operated maintenance operations. Such operations include: Switching opening/closing valves, changing faulty electric boards, drilling and sawing pipes. An operator will control either the robot's spatial translation or its end effectors' position by the use of joysticks.

The robot itself will be extensively described in following sections however a brief introduction is in order. Robots can be classified as follows:

- *Robot Manipulators*
- *Mobile Robots* $\begin{cases} \begin{cases} Wheeled\ Robots \\ Legged\ Robots \end{cases} Ground\ Robots \\ Submarine\ Robots \\ Aerial\ Robots \end{cases}$

(Kelly, Santibáñez & Loría, 2005, p 3).

The robot subject of this thesis finds its place in both groups being a Mobile Ground Wheeled robot possessing two robot manipulators. The classification can be further expanded considering the wheel type and the tracking system used. This robot will make use of a special type of Omni directional wheel called Mecanum wheel.

The other pivotal part of the robot are the manipulators which can be defined as, according to Kelly et al. (2005, p. 4): "A mechanical articulated arm that is constituted of links interconnected through hinges or joints that allow a relative movement between two consecutive links". Such links and joints will be treated individually in a simulated environment and will therefore be mentioned in future sections.

1.2 Research Problem

A variety of solutions exist for a robot's control system: from fully developed and supported ones like LabView, dSpace or TwinCAT to user-made or Open Source like OROCOS (Open RObot COntrol Software) and ROS (Robot Operating System). The latter will be the subject of this thesis and its implementation although in the past other solutions were mostly implemented.

A solution based on TwinCAT was presented by Minxiu, Lining and Yanbin (2012) in their work: "Control system design for heavy industrial robot". A different approach was taken by Teymourzadeh et al. (2013) using National Instrument's LabVIEW to redesign the control scheme of a four-legged spider robot in "Adaptive intelligent spider robot". In 2015 Peppoloni et al. used ROS for robot teleoperation demonstrating the simplicity of the final architecture in their work "Immersive ROS-integrated Framework for Robot Teleoperation".

Robotics has been an extensively researched topic since it began in fully mechanical simple systems until it reached its current state and since the beginning it required a platform known as Artificial Intelligence (AI). In fully autonomous systems it serves as a network of information which binds sensorial data with actuators using its own preprogrammed reasoning of how both inputs and outputs are connected. Our work is slightly different since the reasoning is performed by the human operator driving the machine, however this thesis aims to dwell deeper in the development of this network using a relatively new option known as ROS.

It is challenging for any programmer/engineer to connect together different pieces of hardware into a robot network, even more so if they have different manufacturers, drivers, communication protocols or software compatibility. This thesis will evaluate how accurately ROS can integrate, control and simulate a new custom made robot which results from a complex network of sensors and actuators.

The research community has had an increasing interest in publications focused on or using ROS as shown in Figure 1.2. Publications started to arise one year after its official release during the year of ROS' distro release: Diamondback. Due to ROS' growing popularity and ROSCon the number of publications kept increasing after the release of Hydro.

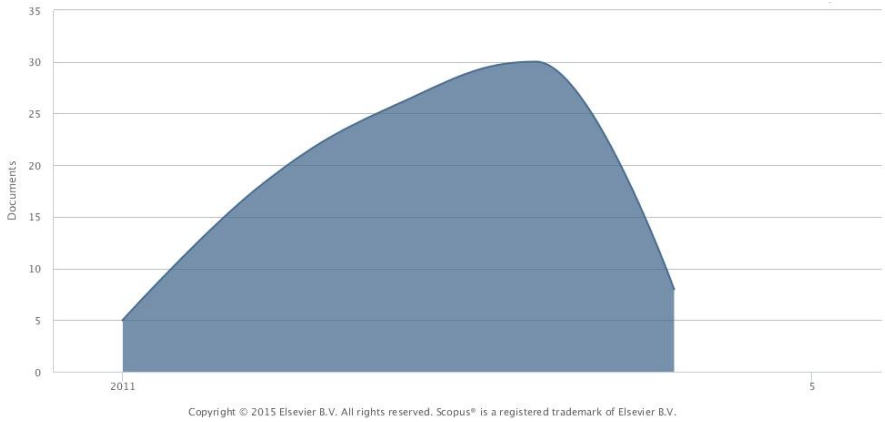

Figure 1.2. Number of publications regarding ROS (Scopus, 2015).

Figure 1.3 on the other hand shows the subject to which the publications pertain being the most popular computer science as expected and followed by engineering. While the computer science publications focus on the development of ROS per se the engineering publications tackle its application in the field of robotics. Both figures were obtained using Scopus' database search engine and statistics plotter.

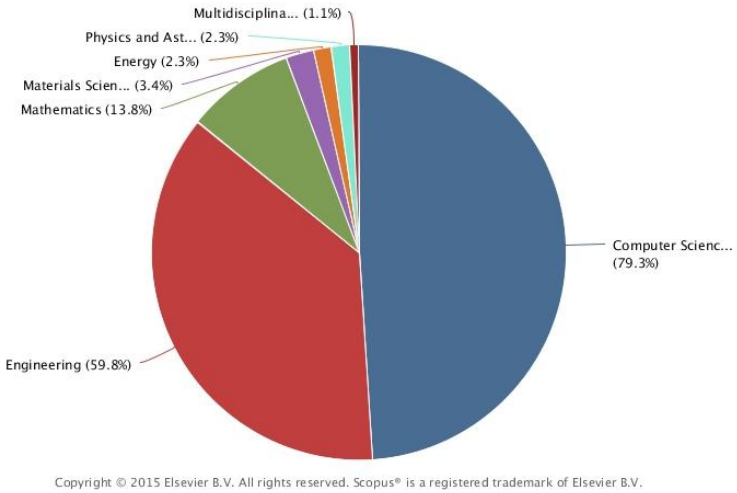

Figure 1.3. Subject of publications regarding ROS (Scopus, 2015).

1.3 Contribution of the Work

This thesis will provide an integrated software solution for a robot performing the tasks mentioned in previous sections. Such solution will make use of an emerging open-source software called Robot Operating System (ROS). After the completion of this work enough knowledge about using the related pieces of hardware, with the previously mentioned

operating system (OS), in our integrated solution will be given as well as a working platform with which already existing algorithms can be improved and new features easily added.

A stability analysis will be performed first theoretically taking into consideration the shifting of the robot's moment of inertia in a Tip-Over study. Afterwards the robot will be simulated in a three-dimensional environment which will further prove its Tip-Over stability and capability of ROS to control such a complex system. The robot will have full awareness of its spatial extension and will prevent self-collisions automatically.

2 CONTROL SYSTEM DEVELOPMENT OF MOBILE ROBOT

First the pieces of hardware that comprise this robot will be introduced. Afterwards the control system will be sketched as well as some insights about the use of this particular operating system. The different parts of the controller layout will be broken down into pieces based on the same criteria used previously and also on the hardware related to it.

2.1 Hardware Layout of the Mobile Robot

The robot is formed by a complex network of computers, sensors and actuators which will be divided into three groups: the driving system, robotic arms and sensor network as shown in Figure 2.1. There is a fourth group which is not integrated in the network but is still a part of the system: extra components. They will be briefly described for the sake of clarity.

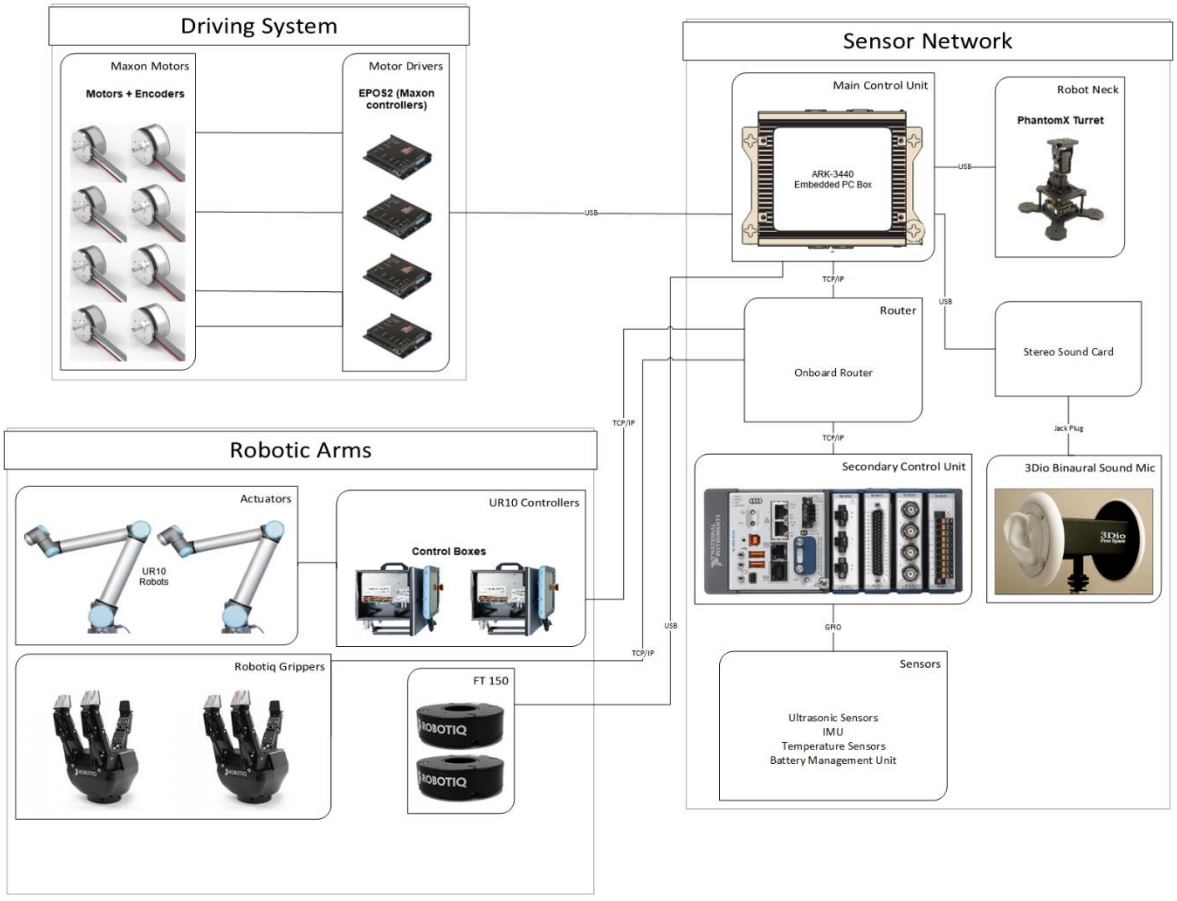

Figure 2.1. Robot Parts: Driving system, robotic arms and sensor network.

2.1.1 Driving System

The driving system consists of four EPOS2 70/10 Maxon controllers connected one to each motor and in series to each other leaving the last one as connection to the main computer. This is shown in the upper left corner of Figure 2.1. Each EPOS2 will transmit and receive data in series using CAN bus except for the last one which will communicate with the main computer through USB.

The necessary cabling for each motor controller is shown in Figure 2.2 where J1 provides the power supply (+11~70VDC), J2 is connected to the motor windings and shield, J3 is the Hall sensor and J4 the encoder. J9 is the USB connection to the main computer although this cable will only be used in the last controller.

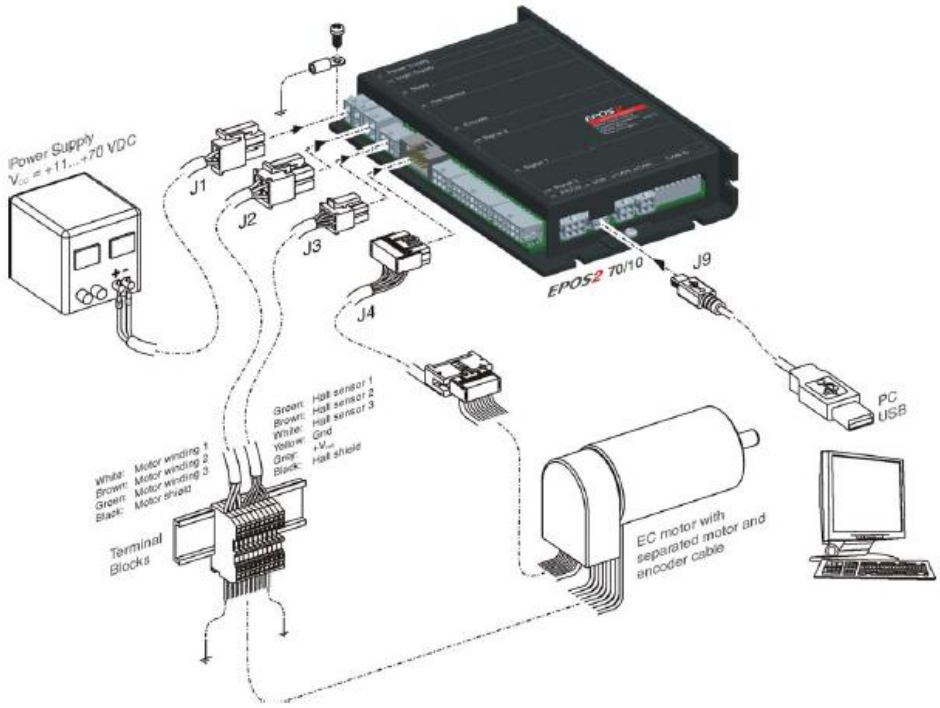

Figure 2.2. Cable layout for each EPOS2 (Maxon Motors, 2013).

The wheels themselves are Mecanum wheels. Mecanum wheels are used in order to achieve omni-directionality which can be defined as the ability to move within a planar space in any direction starting from any possible configuration. This is carried out by the use of rollers located around the wheel's radius which form a certain angle α with respect to the axis of rotation. (Tlale & De Villiers, 2008.)

2.1.2 Robotic Arms

As actuators two UR10s were implemented in a T-shaped structure so that the common workspace was maximized. Two-hand operations can therefore be handled and for that one robotic gripper is attached to the end effector joint of each robotic arm: Robotiq 3-finger grippers. Two of them with their respective force sensor attached.

As shown in Figure 2.1 (lower left corner) each UR10 is connected to its respective control box which is then connected through TCP/IP to the onboard router (and thus to the main computer). A similar layout is used for each robotic gripper bypassing the control box altogether and connecting directly to the router.

One force/torque sensor is used for each gripper as shown in Figure 2.1 which were designed for UR manipulators by Robotiq: FT 150. Such sensors are connected to the main control unit through USB.

The inverse and forward kinematics of both arms are not included in this thesis since they are already taken care of by the software. MoveIt, which is described in later sections, provides inverse kinematics calculations by a simulation of the arm in its own environment therefore simplifying our task. It performs automatic path planning if set up correctly.

2.1.3 Sensor Network

The sensor network comprises the external sensorial data acquisition and processing. It therefore includes sensors and computers plus the neck actuators which are ultimately controlling sensors (cameras).

The main part of the sensor network (and the robot) is the embedded computer: Advantech ARK-3440. This computer runs Linux Ubuntu 14.04.3 LTS with ROS Indigo and acts as the ROS master. It reads information from the onboard router coming from both robot arms and grippers plus the secondary control unit: NI’s compactRIO.

CompactRIO is used as hardware interface allowing very different types of sensors to be used: Ultrasonic sensors for additional spatial information, temperature sensors to monitor the robot’s temperature distribution and one Inertial Measurement Unit (IMU) for navigation

error correction. It uses LabVIEW's ROS package which allows any Visual Interface (VI) to act as a node and post/subscribe ROS topics.

As previously mentioned the robot neck (PhamtomX Turret) is included in this section as it directly controls one of the main sources of sensorial information: the cameras. This component has not been included in Figure 2.1 since it works independently from the abovementioned components although it will ultimately be used in the control room to operate the robot. PhamtomX is controlled through its microcontroller (arbotix) which interfaces with the main control unit through USB.

PhantomX also holds another sensor which helps with the driver's feeling of immersion: 3Dio binaural microphone. Binaural microphones have two different sound receptors ideally separated a distance similar to that of human ears. It creates a stereo sound signal which is described as binaural sound as long as it is played back in a similar way (using earphones or headphones). Binaural sound is another step forward in virtual immersion allowing the human receptor to perform binaural processing which in the words of Grantham (1995, pp. 297-298): "the functions underlying any human capabilities that are rendered possible or superior by the use of two human ears rather than one". Meaning that actions such as determining the position of a sound source in 3D space are possible thanks to binaural sound.

2.1.4 Extra Components

Some components are left out of Figure 2.1 since they do not directly pertain to the scope of this thesis although their importance is discussed here. All of the abovementioned components are crucial parts of the robot although the main sensorial information is not directly included in the network: The vision system.

The vision system consists of four cameras placed in three different locations. The main two cameras are located on top of the robot's neck providing the main source of visual information for the driver. They will be located so that they will produce a three-dimensional video stream which is to be directly interpreted by the operator. The other two cameras will be placed on each arm's end effector so that precise information can be obtained inside the arm's reach

Another important and yet independent part of the robot is one onboard quadcopter. The task to be performed by the quadcopter is to provide one extra live video stream of unreachable places or extra angles.

All live streams are obtained and transmitted independently. Each of them use their own radio transmission channel which is received at the control room and shown at the operator's table. A remote control for the quadcopter will be located at the station while the controls for the neck and robot arms will be performed through ROS.

2.2 Software Development of the Mobile Robot

ROS is what makes the programming and simulation of this robot possible: the main core. The operating system connects every piece of mentioned hardware together and sends the relevant parts to the control room so that the operator can use them to carry out a variety of maintenance operations. This section will tackle the implementation of ROS in each of the parts described in "Hardware Layout of the Mobile Robot". The same separation will be made adding whenever possible a node diagram and specifying to which package they belong.

2.2.1 ROS Deployment of Mobile Robot

A brief description of the concept behind ROS, its basic structure and main parts is conducted here. First the emerging properties of the OS (Operating System) will be exposed continuing with the node/topic structure and members of every ROS network.

a) Network Based Intelligence

A wide variety of ways exist of creating a complex artificial system: from the simplest manner of interpreting input into output to self-learning autonomous intelligence. Every algorithm that interprets and arranges data is similar in such a way that it was programmed by a closed group of individuals which had zero or minimal interaction with the outside world. That leads to an efficient and yet artificial intelligence which is bounded by a small group of individual's capabilities.

The way that nature is "programmed" in some fields is much different. Ant colonies, for example, can achieve very complex tasks like detection and storage of food supplies and

even farming. This may not sound very surprising but the fact that they pretty much have no intelligence at all or no master to give those commands makes it more challenging. They work as a whole very efficiently without any "peacemakers" or individual intelligence relying on the concept of emergence. The term "emergence" was coined by the English philosopher Lewis (1875, p. 412): "Although each effect is the resultant of its components, the product of its factors, we cannot always trace the steps of the process, so as to see in the product the mode of operation of each factor. In the latter case, I propose to call the effect an emergent. It arises out of the combined agencies, but in a form which does not display the agents in action."

What Lewis (1857) tried to explain was that multiple simple systems working in an apparently chaotic manner can achieve complex behavior when working as a whole creating therefore a network based intelligence. The same concept can be applied to the design an AI making it "less artificial". A system which is formed by the shared knowledge of thousands of individuals with completely different backgrounds and aims can somehow be described as an emergent property. Emergence leads to our selected control system, a system which grows and changes on a daily basis due to contributions from researchers all over the world: Robot Operating System.

b) The Operating System

As defined by the official ROS (2015d) website: "The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms."

ROS allows for the division of a complex task such as controlling a mobile robot into multiple smaller tasks with a structure of nodes and topics. A node is a part of the overall program in which a specific set of tasks happen (e.g. reading from a sensor). Such tasks can interact with other tasks in different nodes by publishing or subscribing to topics thus creating a network of information flowing from node to node under the rule of the master (roscore).

Figure 2.3 shows the node-topic structure of one of ROS' basic tutorials. In it two nodes are depicted: "/teleop_turtle" and "/turtlesim" and one topic is being published by the first and

subscribed by the latter. The result is "/turtlesim" will perform a motion described in the topic "command_velocity" which is published by the node "/teleop_turtle". (ROS Wiki, 2015).

Figure 2.3. ROS node-topic structure (ROS, 2015b).

ROS has allowed researchers all around the world to unite without any purpose other than the greater good. Each specialist can focus on creating an interface for any task of their choosing like using a device's specific libraries inside one of these nodes easing greatly the task of the programmer/engineer who finally decides to put many different nodes together. Bugs can be easily tracked and fixed thanks to its increasing community while also implementing suggested improvements in specific parts.

The way a whole network of nodes and topics can be executed at the same time is by the use of launch files. Such files are XML type language and allow for the use of arguments and parameters while executing nodes. An example of launch file can be found in appendix 1. Launch files can execute each other while specifying execution arguments allowing for almost unlimited possibilities of code reuse.

c) Elements of ROS

The main elements of any ROS network will be described here starting from the master, continuing to the parameter server and moving on to nodes and topics. All of these elements work in conjunction to bring the OS to life.

The master (also known as roscore) is a unique entity in charge of managing the publishing and subscription of topics from any node inside the network. Whenever two nodes share the same master there exists bidirectional communication. Inside the master the parameter server can be found. The parameter server is a shared dictionary with which nodes interact at runtime to exchange information. The list of possible elements to store is almost endless but it comprises controller information, robot joint layout, etc. (ROS, 2014a).

As mentioned in the previous section a ROS network is made out of nodes. Nodes are processes in which some computation is performed and the way they communicate with each other is by posting/subscribing to topics. More complex means of communication can be performed by the use of services which allow extra actions like sending a reply once the service has been received.

2.2.2 Driving System in ROS

The driving system is composed of four Maxon's Brushless EC (Electronically Commutated) motor with a 3-channel encoder and a 25:1 reduction gear. Each of the motors is interfaced through an EPOS2 70/10 controller in series as shown in Figure 2.4.

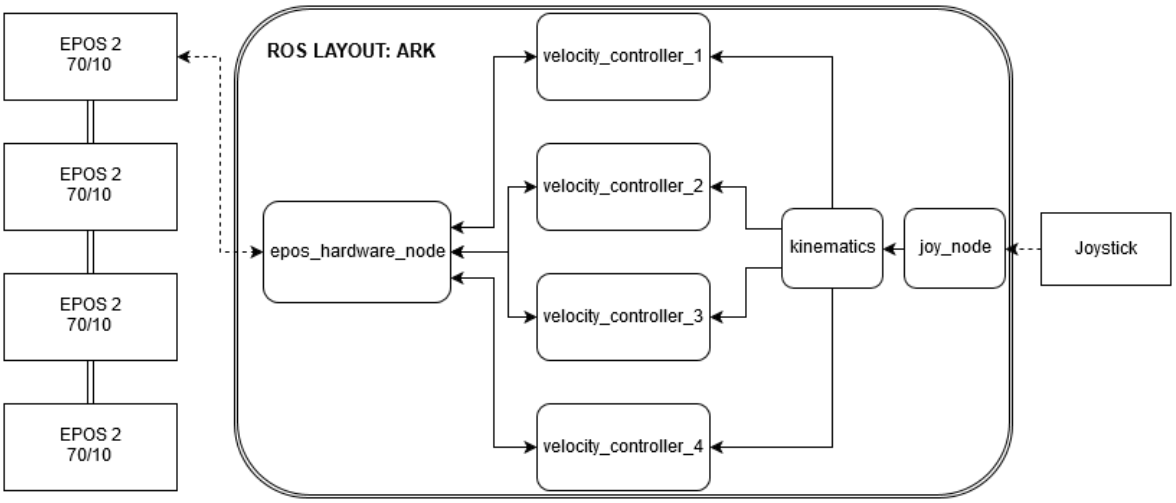

Figure 2.4. Driving system layout.

Figure 2.4 shows the node layout built to control the wheels: to establish a communication between the controllers and the computer an ROS package called "epos_hardware" was used. This package contains a predefined node which establishes the communications with all motors called "epos_hardware_node". This node is connected to a speed controller per wheel which are part of the package "ros_control". Each of the controllers is expecting a speed reference in the form of a topic type "std_msgs/Float64". The reference for each of the controllers is given by the node "kinematics" which was developed by Weiting Le, working in parallel for the same mobile robot. The node "kinematics" is subscribed to the output of "joy_node" which is published from the joystick.

The kinematics of a mobile platform using four mecanum wheels which Center of Mass (CoM) coincides with its geometric center can be described using:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{L+w} & \frac{1}{L+w} & -\frac{1}{L+w} & \frac{1}{L+w} \end{bmatrix}\begin{bmatrix} R_w\dot{\phi}_1 \\ R_w\dot{\phi}_2 \\ R_w\dot{\phi}_3 \\ R_w\dot{\phi}_4 \end{bmatrix} \quad (1)$$

Each wheel's speed is represented by $\dot{\phi}_i$ while v_x, v_y and ω_z are the linear and angular velocities in x, y and z respectively. L denotes the distance between the platform's CoM and the wheel's axis of rotation while w is the distance between the CoM and the axis perpendicular to the direction of rotation as shown in Figure 2.5. The radius of the wheels is R_w and each wheel is numbered from 1 to 4 according to equation (1). (Zimmerman, Zeidis, & Abdelrahman, 2014. pp. 276-277.)

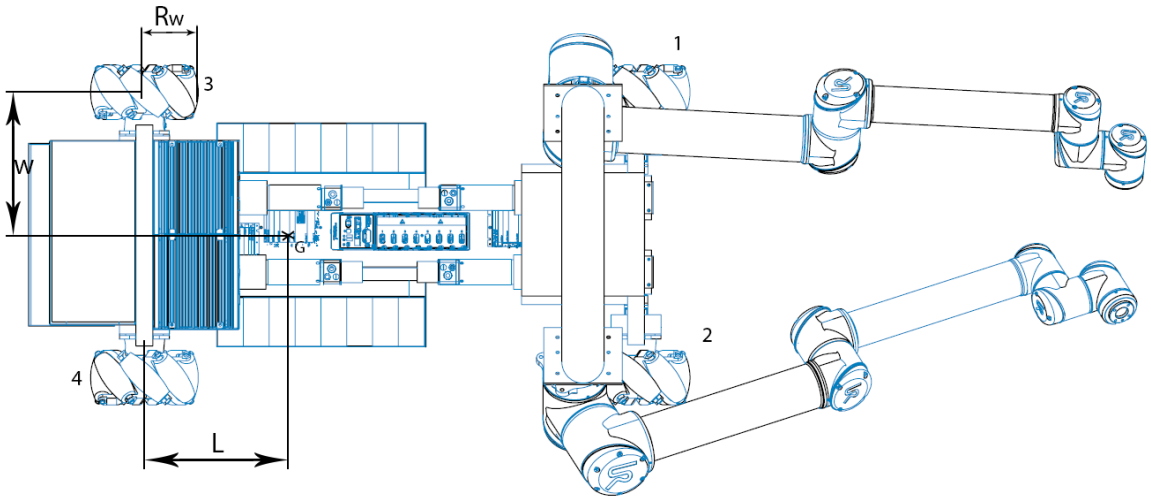

Figure 2.5. Robot's tracking parameters.

By fixing one wheel's rotational speed the other three can be determined using equation (1) and thus the node kinematics can publish those commands into their respective controllers. There are other possible approaches like finding a forth equation to minimize the absolute value of the sum of all rotational velocities but they go outside the scope of this work.

2.2.3 Sensor Network in ROS

This section includes the nodes and topics used to manage sensorial information. A simplified block diagram is used specifying which packages were used.

a) Multi-machine Network

As previously mentioned multiple machines will be used: two onboard computers and a remote computer situated in the control room. All of these computers will communicate by the use of a router which will give each of these machines their own IP address. ROS' node structure was designed keeping this in mind therefore this task can be done by simply using the variable: ROS_MASTER_URI supposing the following conditions are met:

- There is only one master (roscore) running.
- The previously mentioned master is configured using ROS_MASTER_URI for every single node.
- Full connectivity on every port of the bi-directional communication.
- Computers must advertise a name for themselves.

(ROS Wiki, Multiple Machines, 2015).

Before starting a new node in the network the following line needs to be executed: "export ROS_MASTER_URI=http://name_master:11311" with name_master being the name of the computer running the master node. It is advisable to manually give static addresses to each machine. (ROS, 2015a).

Since the robot contains one NI computer a specific LabVIEW package had to be used called ROSforLabview which is being carried out in a google group (ROSforLabview, 2015). As shown in Figure 2.6 the package includes certain VI elements which allow LabVIEW to post and subcribe topics in relation to the same ROS master as the other computers.

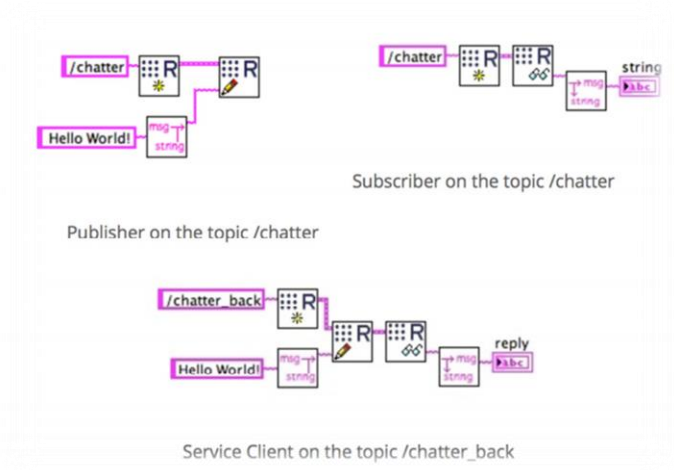

Figure 2.6. ROSforLabview VI elements (ROSforLabview, 2015).

To integrate a LabVIEW's VI into the system ROSforLabview would need to be installed in compactRIO and a simple VI created. This VI would serve two purposes: it would interface with the remaining sensors (Ultrasonic, temperature, etc.) and would post their output into one (or several) topics. Figure 2.7 shows the necessary nodes: LabVIEW's VI is considered a node in this diagram which is connected to sensor_parser in the main control unit through the router since they both share the same master (ROSCORE in Figure 2.7). The information obtained by sensor_parser would be mainly sent through the router (using Wi-Fi) to the control room for another node to display the relevant information for the operator. It was decided to use a sensor_parser node instead of sending the info to the control room directly in case any automatic algorithm based on sensorial information is designed in the future.

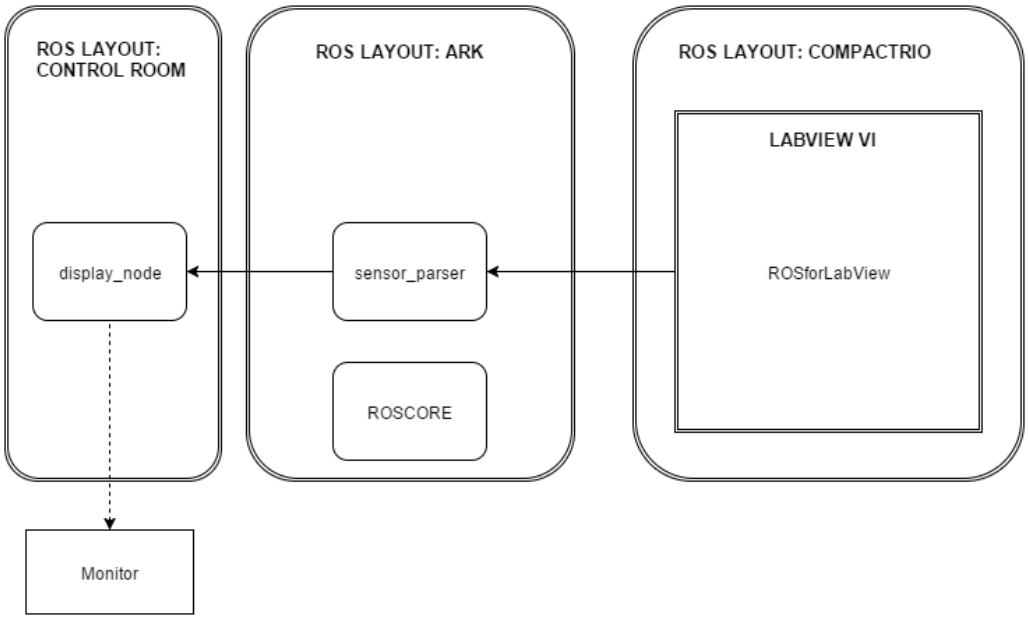

Figure 2.7. ROSforLabView Layout.

b) Robot Neck

To offer mobility in the vision system a pan and tilt turret was installed. The chosen model was initially PhantomX RoboTurret shown in Figure 2.8. The turret is comprised of two Dynamixel servomotors (pan and tilt) and an ArbotiX microcontroller.

Figure 2.8. PhantomX RoboTurret (Trossen Robotics, 2015).

The ArbotiX was programmed using Arduino's Integrated Development Environment (IDE) making use of the ArbotiX library in charge of controlling the servos and also the ROS library in charge of the serial communication with the PC. The program inside the microcontroller awaits the arrival of a "sensor_msgs/Joy" topic and once received actuates both servos accordingly. Figure 2.9 shows how the ROS network is built: a serial node is used to connect any Arduino-like microcontroller like ArbotiX with the ROS master in another device (in this case the Advantech computer). Inside ArbotiX a different node is created which through "serial_node" subscribes to "joy_node".

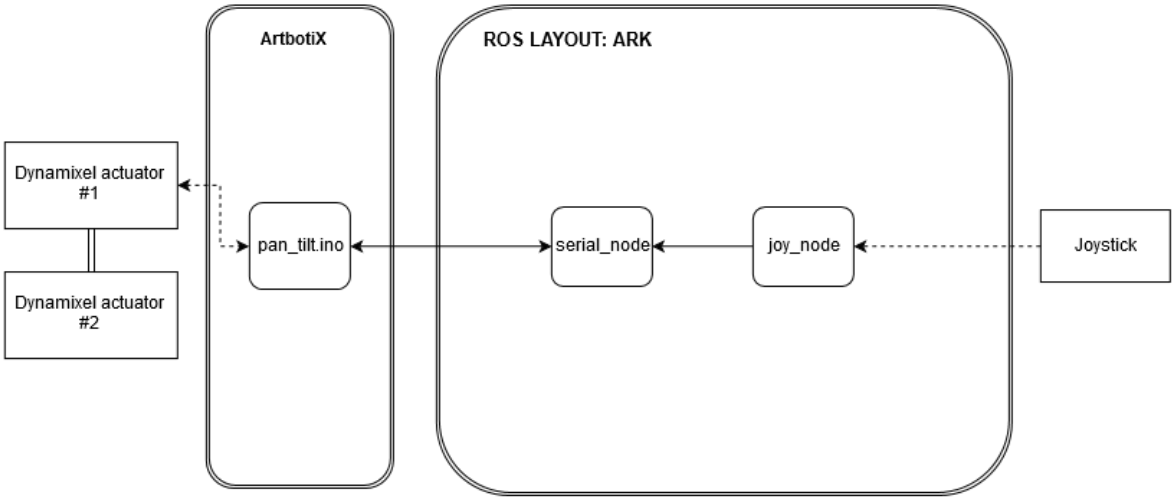

Figure 2.9. Pan and tilt layout.

The microcontroller's code was developed using one of ArbotiX's libraries combined with the serial_node package for Arduino (appendix 2). The microcontroller sets up a node which runs expecting a joy message. When a message is received both servos are actuated accordingly using ArbotiX's function: SetPosition(servo,position).

Unfortunately after some tests it was concluded that the embedded motors were not sufficient for the weight to be supported by the neck and some restructuring was done. The neck structure was redesigned using metal and the motors replaced with higher models. An external brake was attached to the servo controlling the tilt movement. The control layout did not change on ROS' side.

c) Binaural Audio Stream

A simple node network is used for audio streaming which is provided by the package audio_common. Such package offers one node type called audio_capture which will stream an audio signal coming from the default audio recording device. A different node type audio_play will run on a different computer in the control room and due to both computers sharing the same master they will communicate as shown in Figure 2.10 (ROS, 2012).

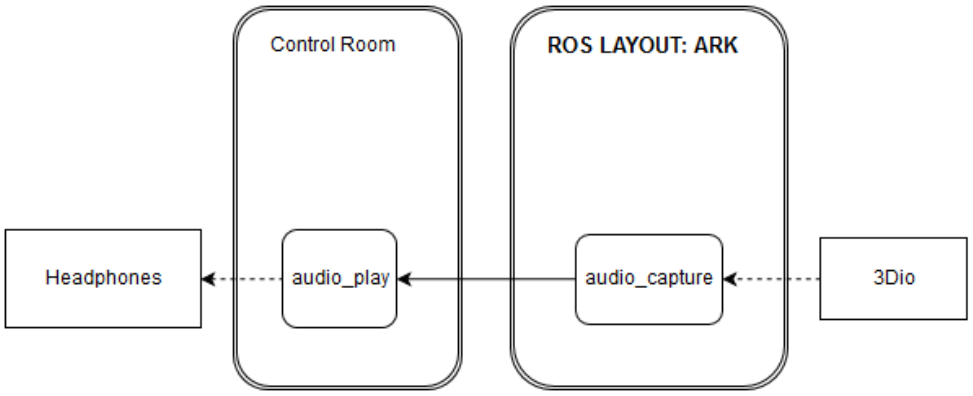

Figure 2.10. Audio stream layout.

2.2.4 Robot Arms in ROS

In order to communicate with both arms the ROS package called "universal_robot" was used. It contains all of the necessary tools to communicate, simulate and control either a UR5 or UR10 (ROS, 2015c).

Although there seem to be multiple ways of controlling the robot arm many specialists agree that MoveIt is the best way. MoveIt is a ROS companion software which takes care of motion planning, manipulation, kinematics and other end effector oriented operations (MoveIt, 2015).

MoveIt creates a C++ class of type MoveGroup which stores and handles all communications between ROS's parameter server, the position and state of all joints and executes whatever trajectories are requested by the user (Sucan, 2015). It can be done either by the use of a programming interface or a Graphical User Interface (GUI) like Rviz. Figure 2.11 accurately describes the role of Movegroup in controlling a robot in ROS. Movegroup will handle both the physical robot and the one in the simulated environment.

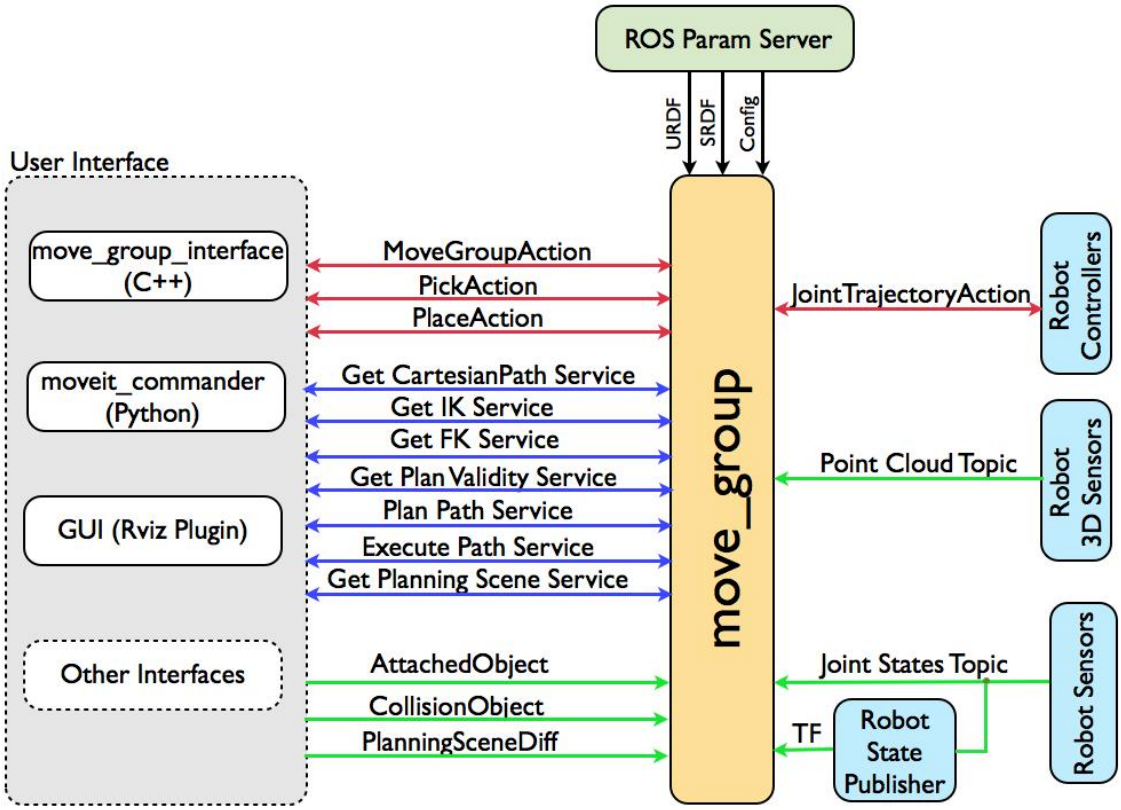

Figure 2.11. Movegroup conceptual diagram (Moveit, 2015).

Once a group has been selected (e.g. left arm) the position of the end effector in Cartesian space can be selected, planned and carried out using a message type geometry_msgs::Pose and MoveGroup's public functions: setPoseTarget(), plan(), setJointValueTarget(), et cetera.

A specific node for teleoperation will be programmed in the final stage of this robot which will read commands from a joystick and translate them into a new goal for each arm to reach in a similar way to the node structure of the driving system and the robot's neck. Figure 2.12 shows the related node diagram. A node called mg_command will send the new position to be reached for each arm to move_group. The node move_group will notify both physical UR10s and execute the path if no collisions were detected. The same will happen in the simulated environment: Gazebo. The robot_state_publisher will be notified and the new joint positions updated.

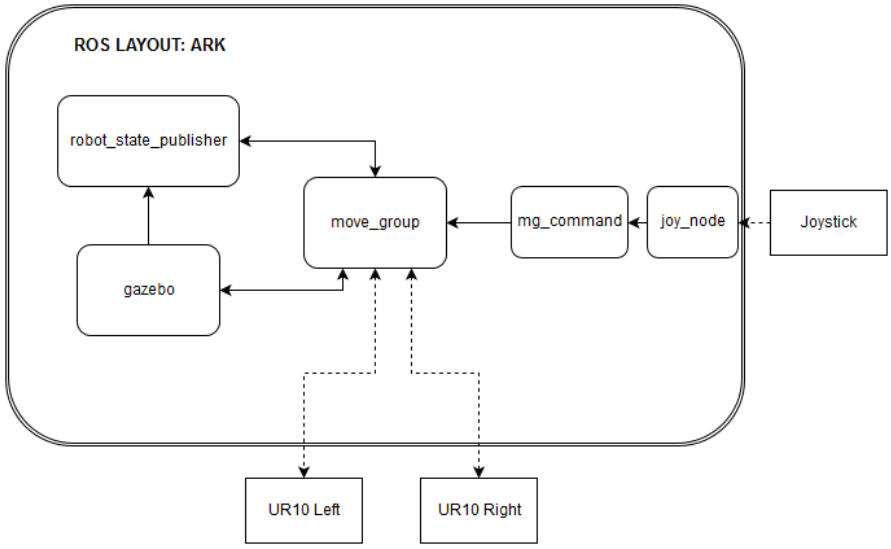

Figure 2.12. Robot arms layout.

Since the robot is still under development the commands will be given simply by using Rviz's GUI in the simulated environment. Rviz is a GUI which allows visualization of virtual robot models stored in the parameter server as well as simulating sensors and their output signals. Rviz can be used with MoveIt which allows for the selection of a compatible position for the arm's end effector in the reachable vicinity. A trajectory can be obtained by planning and in case one has been found (which does not imply collisions) it can be executed.

For the end effector (gripper) and force/torque sensors a different ROS package is needed: robotiq. This package contains three different nodes which are shown in Figure 2.13. These three nodes perform a different task each:

- s_model_tcp_node_i: Acts as a driver reading information from the gripper and/or force/torque sensor. There are two and each of them reads data from a different gripper based on their IP address.
- s_model_simple_controller_i: Sends commands to the driver node. The default node offers simple actions like opening and closing the hand but it can be customized to send more complex actions based on joystick input.
- s_model_status_listener_i: Reads data from the driver node and provides it as output checking the actual status of the gripper. This info can be used by the controller to close the loop.

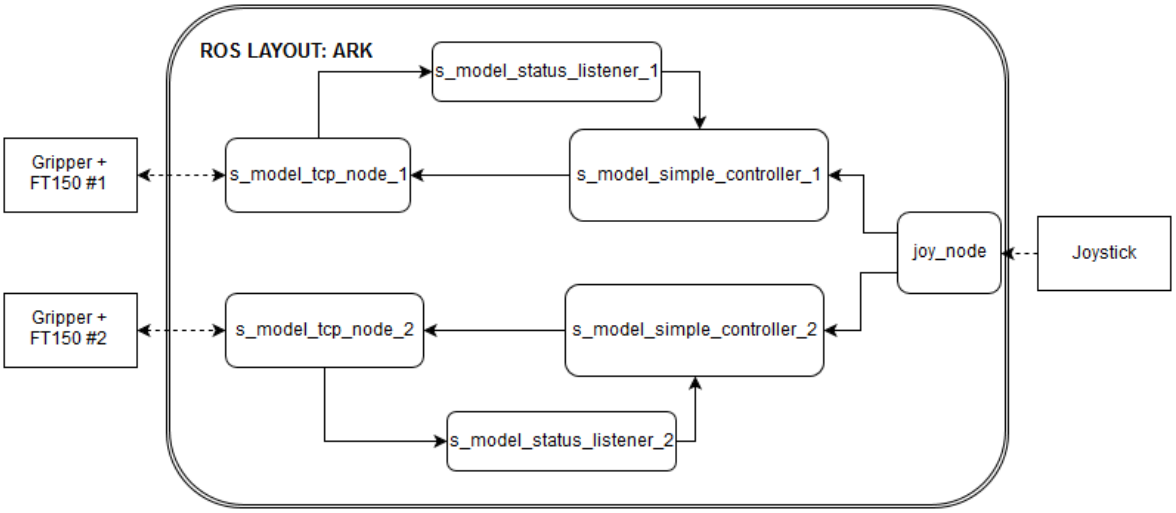

Figure 2.13. Robot grippers' layout.

2.2.5 Complete Node Layout

Although the network has been laid out component by component a complete system graph in depicted in Figure 2.14 which might help comprehend it further. The figure contains every single node previously introduced in sections 2.2.1-2.2.4 which are interconnected when necessary and travelling between different computers.

The main central box called "Main Control Unit" contains most of the network's nodes: driving system (upper), arms (central), grippers (lower), force/torque sensors (lower) and some data acquisition (lower right). On the left side of the figure all the onboard hardware is located (rectangular boxes) while on the right secondary computers are depicted in addition to hardware located on the operator side.

Since the control room is still under development the device used to control the robot remains unclear. Therefore it was decided to assume everything would be controlled by a single joystick as shown in Figure 2.14 (upper right corner). It is likely that the final design will have more than one joystick and therefore more than one joy_node.

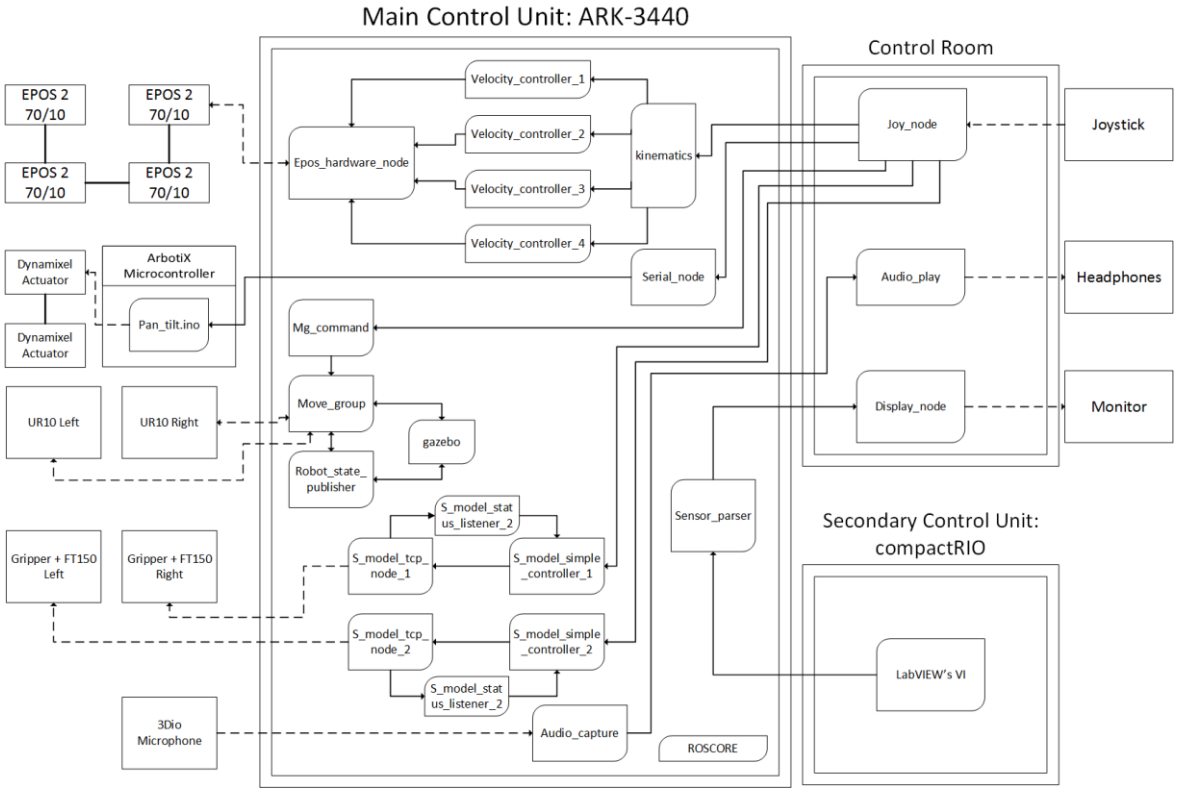

Figure 2.14. Complete ROS Layout.

3 STABILITY AND SIMULATION ANALYSIS OF MOBILE ROBOT

After the main parts of the operating system have been designed and explained some analysis will be carried out. At first the platform's stability is put to the test with a tip-over analysis done in parallel with *Matlab* by taking into account the robot's weight distribution, contact points and end effector position. Secondly a physics simulation will be done to test the robot's capability of moving as an Omni wheeled platform and perform dual arm motions avoiding self-collisions.

3.1 Tip-Over Stability Analysis

This section goes over the analysis performed to ensure the robot's static and dynamic stability. At first a *Matlab* code was developed based on the index for tip-over stability margin defined by Papadopoulos and Rey (1996) in "A New Measure of Tipover Stability Margin for Mobile Manipulators". As shown in Figure 3.1 the angle (θ_i) of the resultant force (f_r) with each of the contact point axes ($\hat{a}_i$) is calculated. If any of the calculated angles turns out smaller of equal than zero the system will tip-over around that axis.

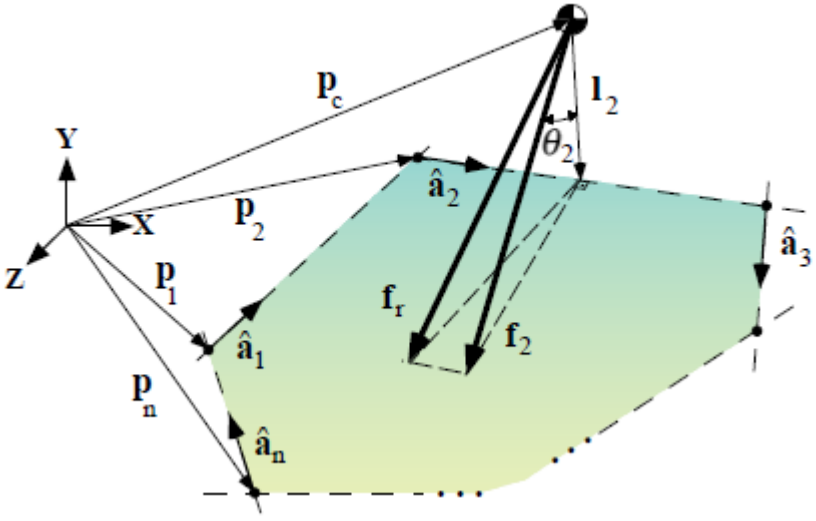

Figure 3.1. Stability Margin Index (Papadopoulos, 1996, p. 3).

The robot's real dimensions and properties were introduced and the stability margin was obtained in different scenarios: hand positions, acceleration, and hand's height using the following equations (Papadopoulos, 1996, pp. 3-4):

Normalized tip-over axes $\hat{a}_i$ and its normal I_i are calculated as follows being p_i a contact point between the ground and the manipulator:

$$\hat{a}_i = \frac{p_{i+1}-p_i}{|p_{i+1}-p_i|} \quad i = \{1, \dots, n-1\} \tag{2}$$

$$\boldsymbol{I_i} = \left(\mathbf{1} - \hat{a}_i \hat{a}_i^{\boldsymbol{T}}\right)(\boldsymbol{p_{i+1}} - \boldsymbol{p_c}) \tag{3}$$

The net force (without taking the ground's reaction force into account) and moment acting on the center of mass are f_r and n_r respectively. To obtain the component acting upon the i-th axis the following equations are used:

$$\boldsymbol{f_i} = \left(\mathbf{1} - \hat{a}_i \hat{a}_i^{\boldsymbol{T}}\right)\boldsymbol{f_r} \tag{4}$$

$$n_i = (\hat{a}_i \hat{a}_i^T) n_r \tag{5}$$

$$f_i^* = f_i + \frac{\hat{I}_i \times n_i}{|I_i|} \tag{6}$$

Where f_i^* is the combination of the force component and the force couple equivalent to the net moment component. To obtain the stability margin θ_i is calculated as an angular margin and is transformed into α_i which is a force margin:

$$\theta_i = \sigma_i \cos^{-1}(\hat{f}_i^* \cdot \hat{I}_i) \tag{7}$$

$$\sigma_i = \begin{cases} +1 & if\ \left(\hat{I}_i \times \hat{f}_i^*\right) \cdot \hat{a}_i < 0 \\ -1 & else \end{cases} \tag{8}$$

$$\alpha_i = \theta_i |f_r| \tag{9}$$

Each θ represents the angle between its corresponding contact point axis and the net force going through the CoM as shown in Figure 3.1 (θ_2 is depicted). It can be concluded then that the bigger each θ is the higher the margin before tipping-over around that axis. Whenever any of the angles reaches zero the system will lose a contact point in the opposite end and a tip-over accident will begin. A negative angle (if it ever came to happen) would indicate that the net force's absolute value and direction will without a doubt cause the system to tip-over around that axis.

The system was translated into the robot as shown in Figure 3.2. Figure 3.2 depicts the same angle as Figure 3.1 but with only four contact points between the system and the ground. All four tip-over axes are shown in red while vectors P1…P4 are in blue. In Figure 3.2 each P represents one contact point between the robot and the ground, although it would be more realistic to have contact lines instead of points the simplification was made so that calculations would be simpler. The robot is modelled using spherical angles as shown in

Figure 3.3. The shoulder angles of the other arm would be equivalent but were left out for the sake of clarity (θ_{21} & ϕ_{21})

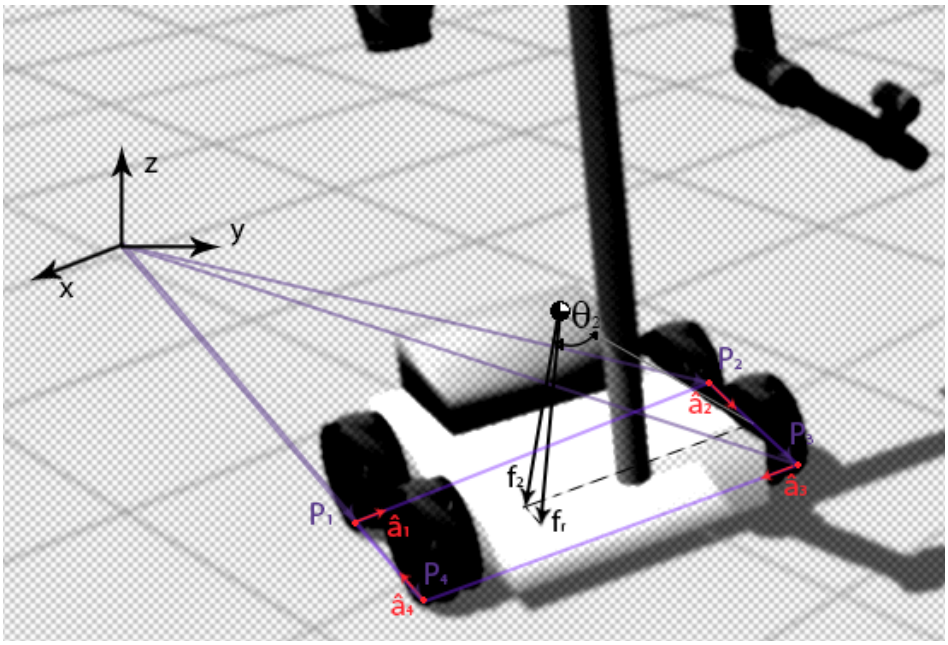

Figure 3.2. Stability Margin Index of the Maintenance Robot.

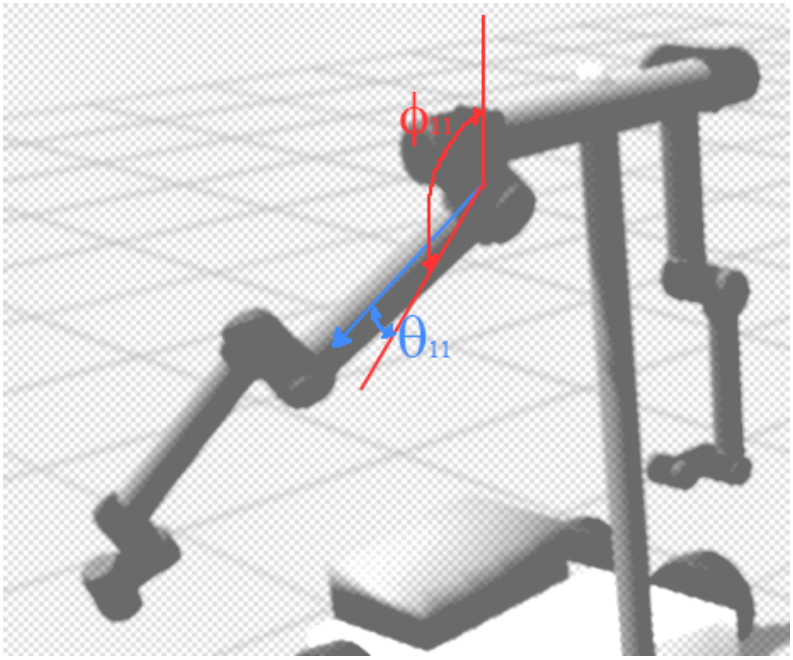

Figure 3.3. Robot's angles.

The equations were introduced in *Matlab* and the following results obtained: Figure 3.4 shows the way in which the position of the arms reflects on the stability of the system. The

angles are slightly different (they have a 45º offset) to better reflect the dynamic behavior of the angles since ideally equal angles results in a constant margin for each angle. Theta and phi reflect respectively the spherical coordinates and as can be observed in Figure 3.4 an increase in both phis ($\phi = 0$ corresponds to the arm pointing upwards and $\phi = \pi$ downwards) results in a bigger margin. For each phi as thetha increases the system becomes less stable which make sense since the arms are shifting the CoM (Center of Mass) outside of the contact area with the ground.

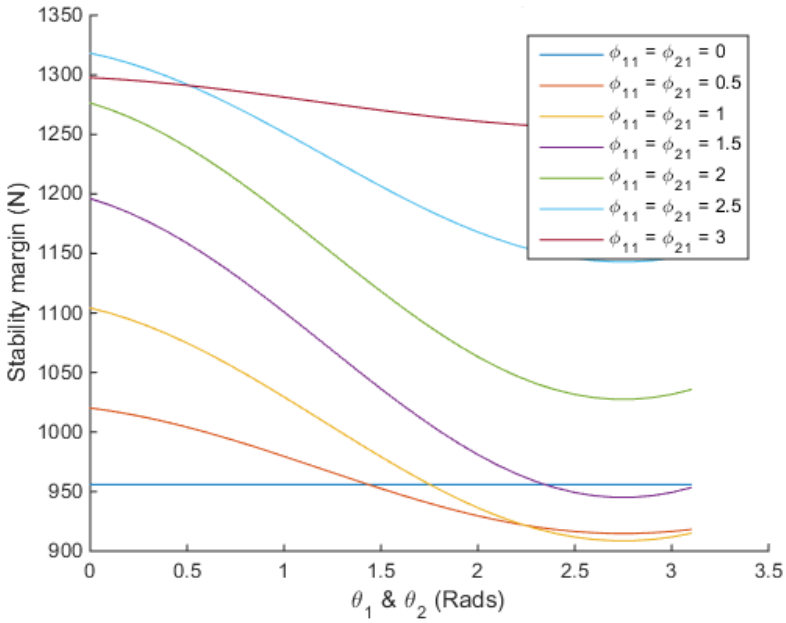

Figure 3.4. Effect of the first angles.

The position of the arms in global Y with respect to the geometrical center of the chassis is set to 0.8465 m although Figure 3.5 shows the effect of this parameter in the stability. As expected the conclusion drawn is that bigger heights result in a smaller margin.

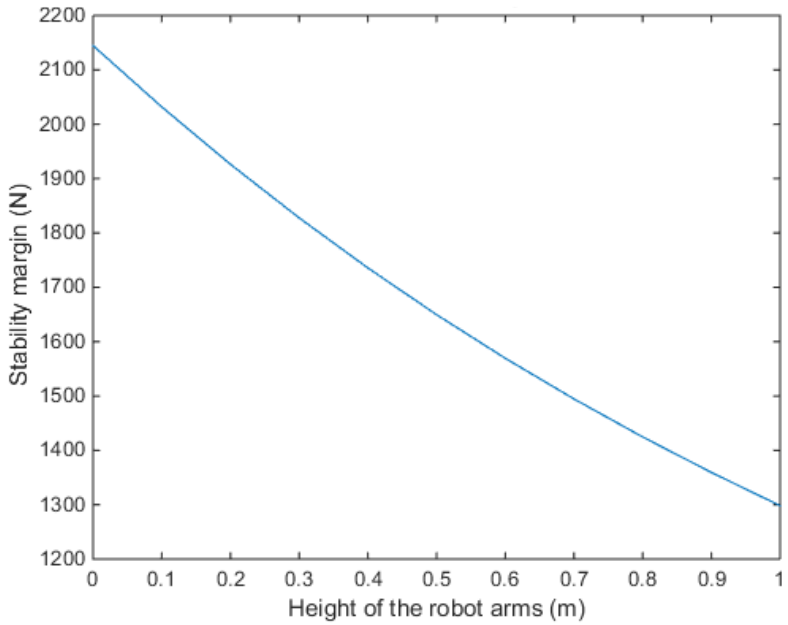

Figure 3.5. Effect of the arm's height.

Figure 3.6 shows the effect of one of the forces acting upon the system: inertia. The effect of increasing the system's acceleration is favorable until reaching a peak near the area of $6\ m/s^2$ after which it starts to decrease linearly. This is due to the fact that the system has four possible tip-over axes and the margin is defined by the one which angle is smaller, the increasing inertial force increases one of them while decreases another and thus creates the peak.

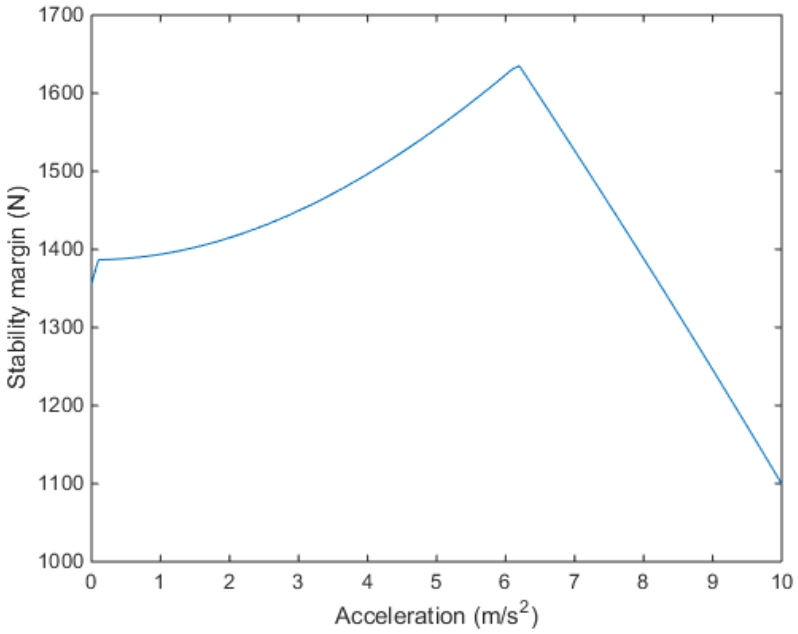

Figure 3.6. Effect of the acceleration.

The obtained data will be compared and confirmed using a working virtual model of the robot with a simple teleoperation algorithm.

3.2 Simulation of the Robot in ROS

ROS' unique control network relies on a virtual model of the robot stored on the parameter server which is ideally tied to actuators and sensors the real robot possesses. The same way the class MoveGroup relates both virtual and physical UR10s a file describing all relations and joints the robot might have can be built: Unified Description Robot Format (URDF).

3.2.1 Unified Description Robot Format

An URDF file is written in XML format and is based on the description of simple components which are tied to others by moveable or fixed joints (ROS, 2014c).
These objects are known as links and may possess the following attributes:

- Inertial: The origin and orientation of the inertial reference frame can be defined as well as the link's mass and 3x3 inertial matrix.
- Visual: An origin can also be defined as well as a geometry element which can either be a box, cylinder, sphere or any irregular mesh. The material can also be defined which will only change the visual properties based on it.
- Collision: Similar to the visual element in many fields but usually simplified. Collisions will be calculated based on the geometry element chosen here therefore a simple element is preferable to a complicated mesh to reduce calculation times.

(ROS, 2014b).

The relation between links is established by joint elements. Joint elements define parent to child relations which result in the kinematic chain known as the robot. A link element which has no parent element is known as the root link. The types of joint elements are described below:

- Revolute: rotational joint along a user defined axis which has a lower and upper limit.
- Continuous: a revolute joint that can rotate in any direction continuously.
- Prismatic: sliding joint which allows displacement along the defined axis. Has upper and lower limits.
- Fixed: does not allow any type of movement.
- Floating: allows all six degrees of freedom.

- Planar: motion along a plane perpendicular to the given axis.

(ROS, 2013)

3.2.2 Simulation Software

As an earlier stage in development it was decided to study the robot first in a simulated environment. A simulation will provide some results about the stability of such a system as well as useful data necessary for teleoperation. To carry out the simulation it was decided to use Gazebo, a physics engine which started as a part of ROS and is now its own independent software. It offers simulation of dynamics displayed as realistic 3D graphics of some already made robot models and parts to which virtual sensors and plugins can be added. Gazebo can be easily integrated with ROS by the use of the parameter server and the package ros_control. (Gazebo, 2014.)

Gazebo works as any other ROS node exchanging information with the parameter server and the robot state publisher. For this thesis the physics of the mecanum wheels were simplified with gazebo's plugin: planar move.

Planar move simulates the behavior of an Omni-wheeled platform by subscribing to a topic called "cmd_vel" of type "geometry_msgs/Twist" and executing that command. The message to be published is formed by two linear velocities: v_x and v_y as well as an angular velocity along the z axis: ω_z. It is then simulated in the gazebo environment as a displacement of the robot's base in the manner described by the velocity vector.

For testing purposes and time constraints the robot model was further simplified to its main components, the Omni-wheeled platform and both UR10 robot arms. It was concluded that for a tip-over analysis and teleoperation testing the former simplifications were accurate.

3.2.3 Simulation Model

A URDF file was built for the tests which will be described in this section. Such file serves as the model for both Gazebo and Rviz, as mentioned in previous sections. Figure 3.7 shows the resulting model used for this section of the thesis which will be detailed bellow.

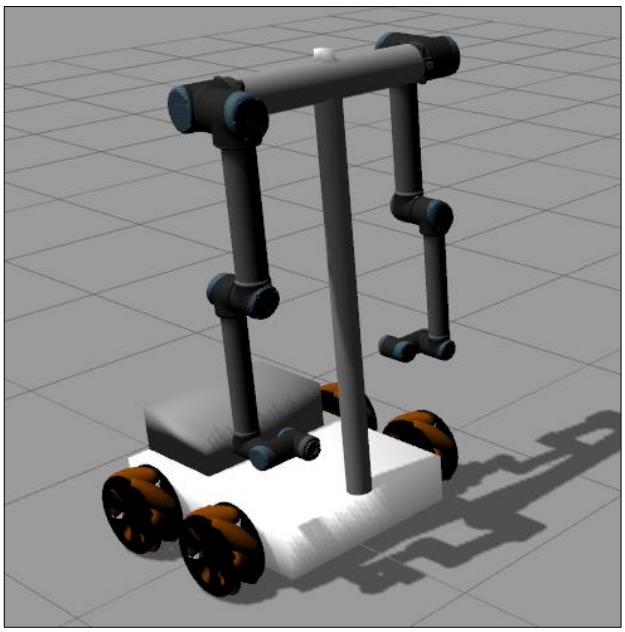

Figure 3.7. Resulting 3D model in Gazebo.

The URDF file establishes the relationships between all of the robot parts (links) included in the simulation by means of joints as shown in Figure 3.8. Figure 3.8 was automatically generated using the function “urdf_to_graphiz”.

The defined root link is called “world” (top-most element in Figure 3.8) and it is not strictly necessary since it does not have a physical entity tied to it but together with “footprint” it allows for gazebo’s planar plugin to work. The real robot starts from the link “odom”, the robot’s chassis, which is built as a simple box. This box is then connected to each of the wheels: left_back_wheel, left_front_wheel, right_back_wheel, right_front_wheel. Every singular wheel is described as a cylinder in its inertial and collision properties using for the visual element a matching model of a mecanum wheel as shown in Figure 3.7. The former simplification was made due to the fact that the planar move plugin was being used. The joint element connecting wheels and chassis was defined as a continuous link to simulate the continuous rotation of each wheel.

Two other parts were connected to the main chassis using a fixed joint: a counter weight (counter_weight) and the T-joint (v_t and h_t). The counter weight is in all effects a box which was only used to compensate the weight of both arms. The final weight distribution and sizes were still undecided at this point so a simplified model was used for simulation

purposes. The T-joint on the other hand serves the purpose of holding both arms in position. It is formed by two cylinders joint together in a T-shaped vertical structure as shown in Figure 3.7.

Three independent elements are connected to the horizontal bar of the T-joint: both UR10 arms and a virtual camera. The camera (camera_link) is joined as a fixed element and will have a gazebo plugin attached called “ros_camera”. Both arms are equal in joint types and link elements all over their structure differing only in their name’s prefix: left and right. Each arm is joined at their base (rightbase_link and leftbase_link) to the T structure with a fixed joint. The model used for each arm was obtained from the previously mentioned package universal_robot which contains models for both UR10 and UR5 easily that can be easily integrated into any URDF.

Figure 3.8. Graphic representation of the robot's URDF file.

Each of the UR10 arms in Figure 3.8 continue to join their respective shoulder_link, upper_arm_link, forearm_link, wrist_1_link, wrist_2_link, wrist_3_link and ee_link being the latest the joining link in case an end effector is used. For further stages of the project the 3-finger gripper from Robotiq will be used. The joint element used for every degree of freedom is revolute the number being six in total.

After the URDF file was successfully parsed MoveIt's setup assistant generated a package with some launch files that assist in controlling the MoveGroup classes defined in it. To start the simulation the following files were executed:

- Centaur.launch: Centaur being the tentative name of the robot during the early stages. This file uploads the URDF file to the parameter server, starts gazebo, spawns the URDF file in it and starts the controllers necessary for every joint.
- Move_group.launch: This file is automatically generated by MoveIt's setup assistant and takes care of setting up and launching the MoveGroup node/action server.
- Moveit_rviz.launch: In charge of launching Rviz and selecting all of the configuration parameters necessary for MoveIt to be integrated.

Another node was added (appendix 1) although it remains commented: centaur_planner. Centaur_planner sends a specific trajectory to the MoveGroup class and is then executed however for testing purposes it was decide to simply use Rviz's GUI.

Figure 3.9 shows the automatically generated diagram of nodes and topics involved in the simulation by using rqt_graph. The resulting figure has a similar layout to that of Figure 2.12 in section 2.2.4 since as it was previously mentioned the simulated and physical environments were designed to be controlled in parallel. The node "fake_joint_calibration" becomes necessary since there are no physical UR10 devices connected to the network and MoveIt will try to synchronize them. The controller spawners are only in charge of creating all the controllers since they are not considered or treated as nodes.

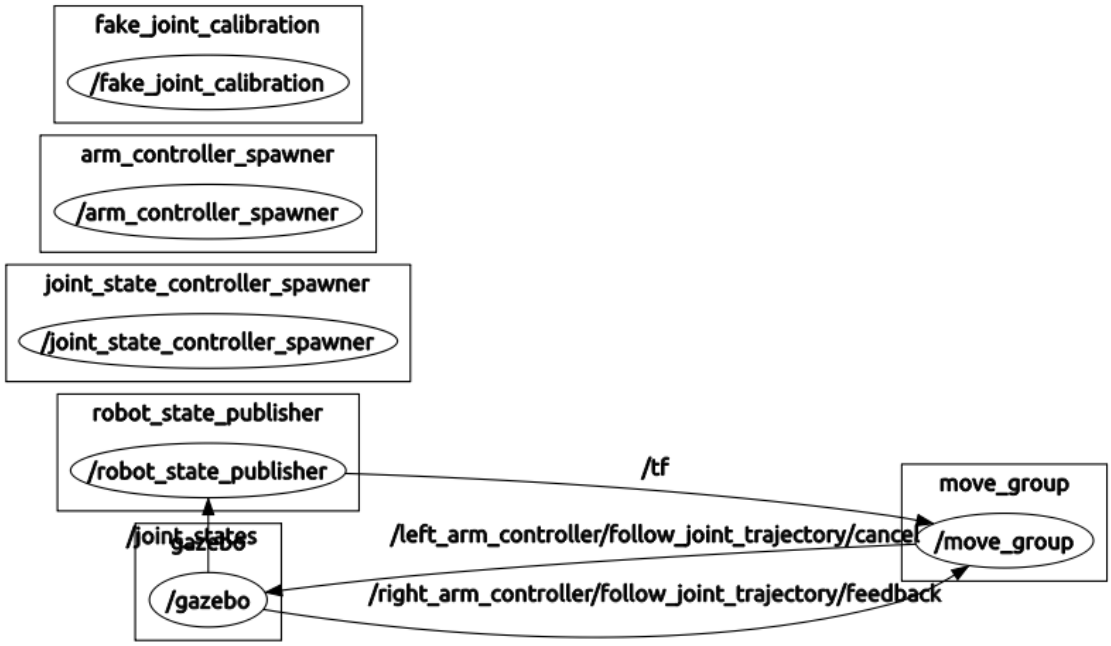

Figure 3.9. Nodes and topics of the simulation.

An alternate version of Figure 3.9 is shown in Figure 3.10 in which the planner_node (appendix 1) was uncommented and used to send several trajectories. As mentioned in section 2.2.4 this will be the way in which the final network is implemented so that new trajectories can be acquired directly from the joystick.

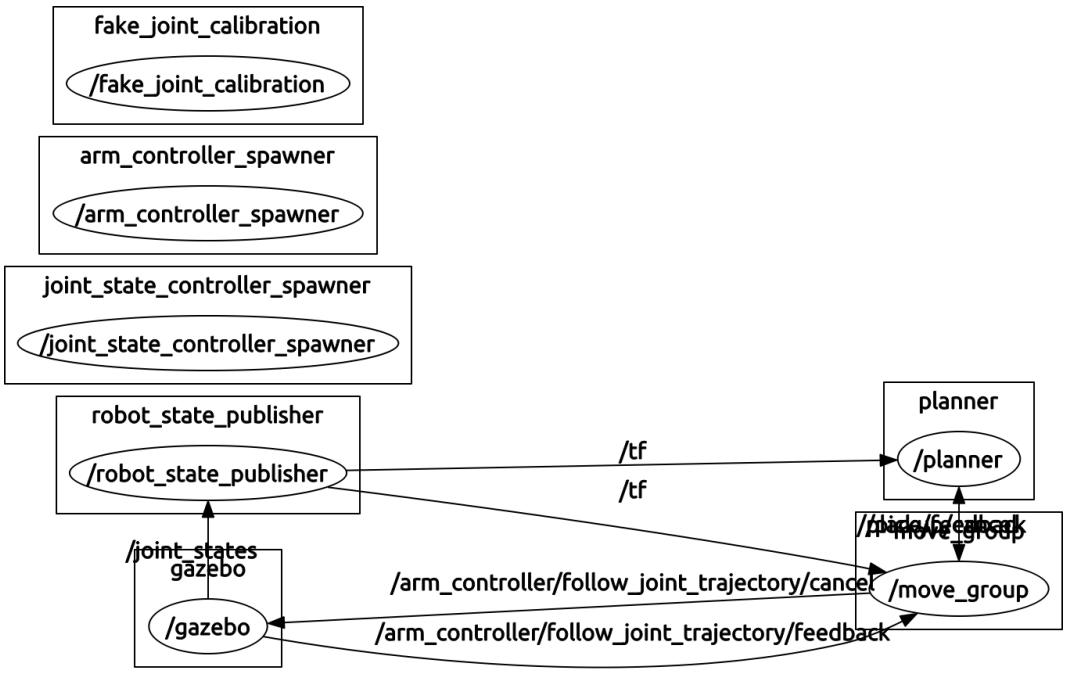

Figure 3.10. Nodes and topics of the simulation using a planner node.

Using Rviz's GUI, as shown in Figure 3.11, the MoveGroup can be selected (either left or right arm) and its end effector's final position can be planned by moving the light blue sphere shown in both sides of the figure. In the left side of Figure 3.11 the initial position of the robot arm is depicted while on the right side a trajectory is being executed. It can also be seen on the left side of Figure 3.12 since both Rviz and Gazebo are synchronized by the robot state publisher.

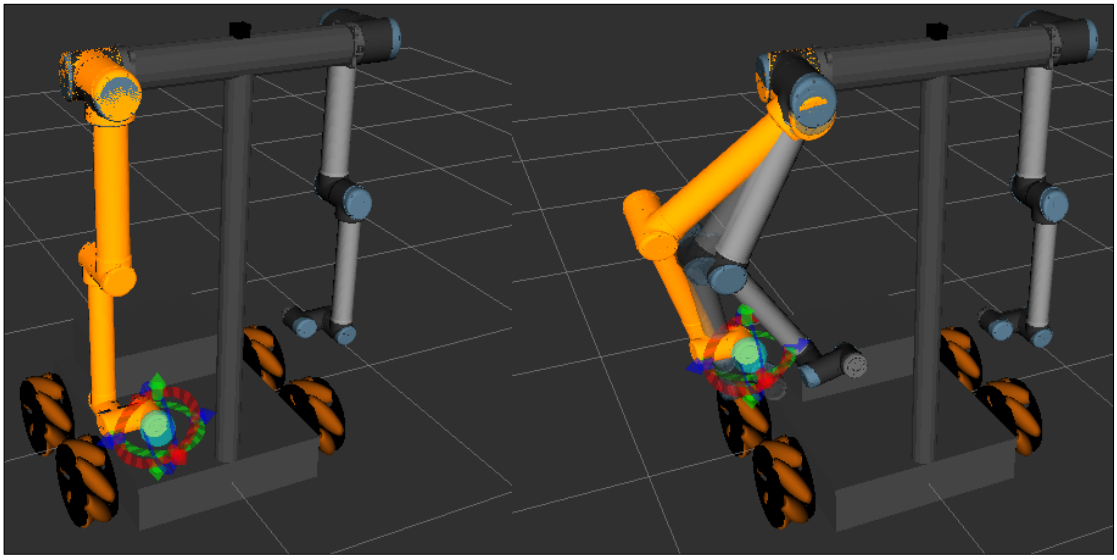

Figure 3.11. Rviz planning: first arm.

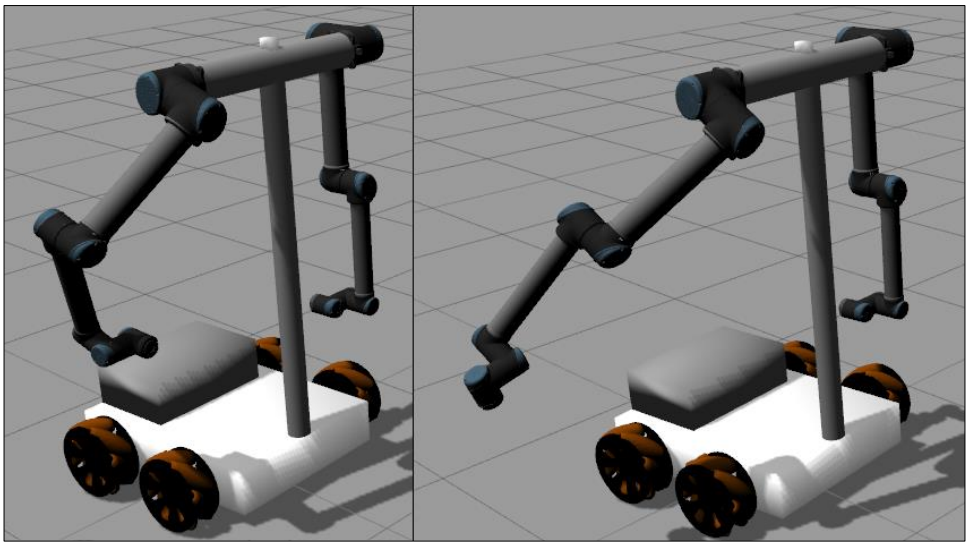

Figure 3.12. Gazebo Simulation: moving first arm.

Figure 3.11 shows on both sides two different trajectories that were planned on Rviz and then executed on Gazebo simulator. Both trajectories were planned without any obstacles

and were therefore executed smoothly. Figure 3.13 on the other hand shows a more complicated path planned: one arm was placed in the front upper part and then the other arm was commanded to reach the same position. As depicted in Figure 3.13 the trajectory is incompatible with the robot's geometry therefore cancelling the path execution.

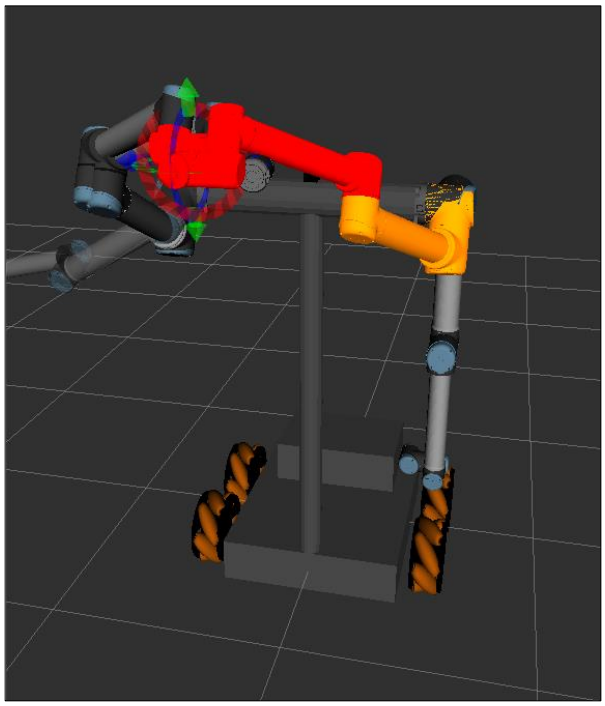

Figure 3.13. Rviz planning: collision detection.

3.2.4 Simulation Results

In order to determine the controller's performance a series of simple test were performed. Each of the UR10 arm is controlled in every joint using a PID controller with a clamping terms which parameters are included in Table 3.1.

Table 3.1. Parameters of one arm's PID controller.

	K_p	K_i	K_d	$i_{clamping}$
Shoulder_pan_joint	600	500	100	100
Shoulder_lift_joint	1500	500	100	100
Elbow_joint	1500	500	100	100
Wrist_1_joint	100	0	0	0
Wrist_2_joint	100	0	0	0
Wrist_2_joint	100	0	0	0

Every wrist joint is simply controlled by a proportional gain while the rest have a much higher proportional gain combined with integral and derivative terms. Any joint having an integral term is compensated by the use of the clamping term which is used to stop and prevent integrator windup. Such phenomena takes place whenever there exists a saturation limit in the signals (practically any real system) and an integral term is used. The result is higher overshoots and settling times which should therefore be avoided. In order to prevent integrator windup a conditional integration is used as described by Visioli A. which stops the integral term whenever a certain limit of the signal is reached ($i_{clamping}$ in Table 3.1) (Visioli, 2006. pp. 37-38).

Three of Table 3.1's parameters were iterated to determine the optimality of their values: the proportional gains of shoulder_pan_joint, shoulder_lift_joint and Elbow_joint. The values previously shown were used for the first test. The initial position was the same as shown in Figure 3.5 after which both arms were asked to travel 0.5 m across positive Cartesian z-axis. The final position of each test is shown in Figure 3.12 where the first test is depicted in the left-most position. For this set of tests only one arm was iterated so that the difference between them would become more apparent.

Figure 3.14. Simulation final positions.

Figure 3.15 shows the end effector's position in global z-axis over time. Since the overshoot is minimal a rough estimate of rise time will be taken into consideration to value the controller's optimality. The approximate rise time of the first simulation is: $t_r = 3.5\ s$ while no other fenomena was observed. The signal settled accurately in both arms using default controller values.

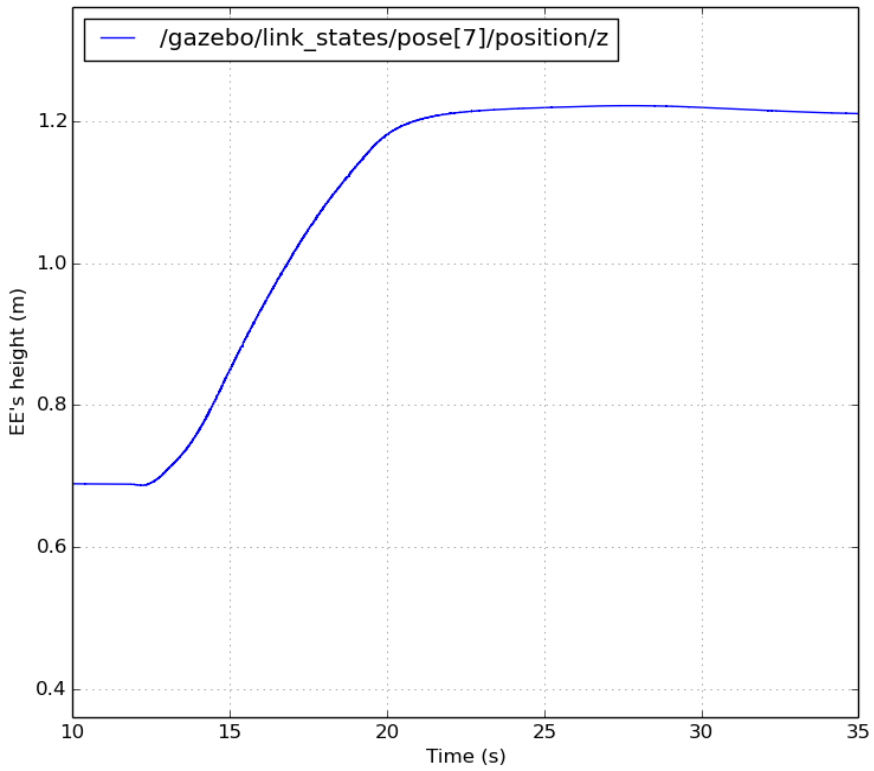

Figure 3.15. Simulation results #1.

The shoulder_pan_joint's gain was increased from 600 to 900 after which the same test was repeated. The robot's final position is depicted in the middle frame of Figure 3.14 and is apparently indistinguishable from the first test. The same variable is represented over time in Figure 3.16 while its rough estimate of rise time is: $t_r = 4.7\ s$ slightly worsening the initial results. It is also apparent that settling time increased as well in the execution of the second test.

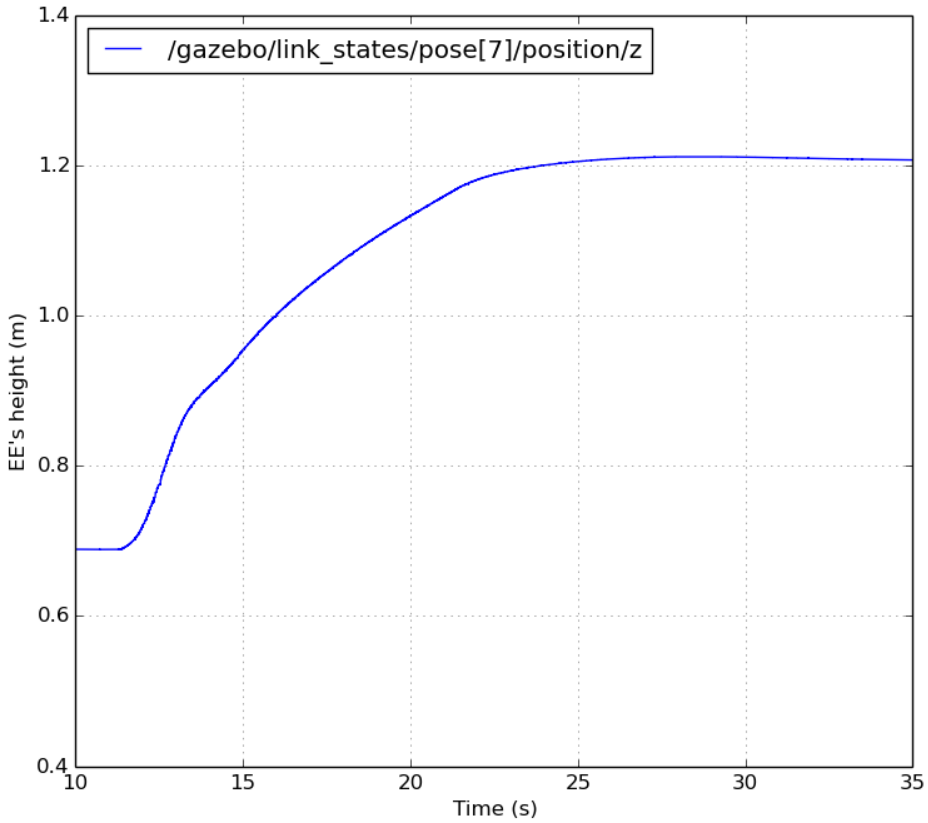

Figure 3.16. Simulation results #2.

The last test was performed further increasing the gain to match up with the other joint gains reaching a value of 1500. The resulting response is depicted in Figure 3.17 sharing a similar shape as both other graphs. The estimated rise time was: $t_s = 5\,s$ which indicates (estimation errors aside) that further increasing the gain leads to least optimal controllers. As shown in Figure 3.14 (right-most figure) a slight z-offset appeared over time which might indicate that the clamping term needs to be readjusted appropriately.

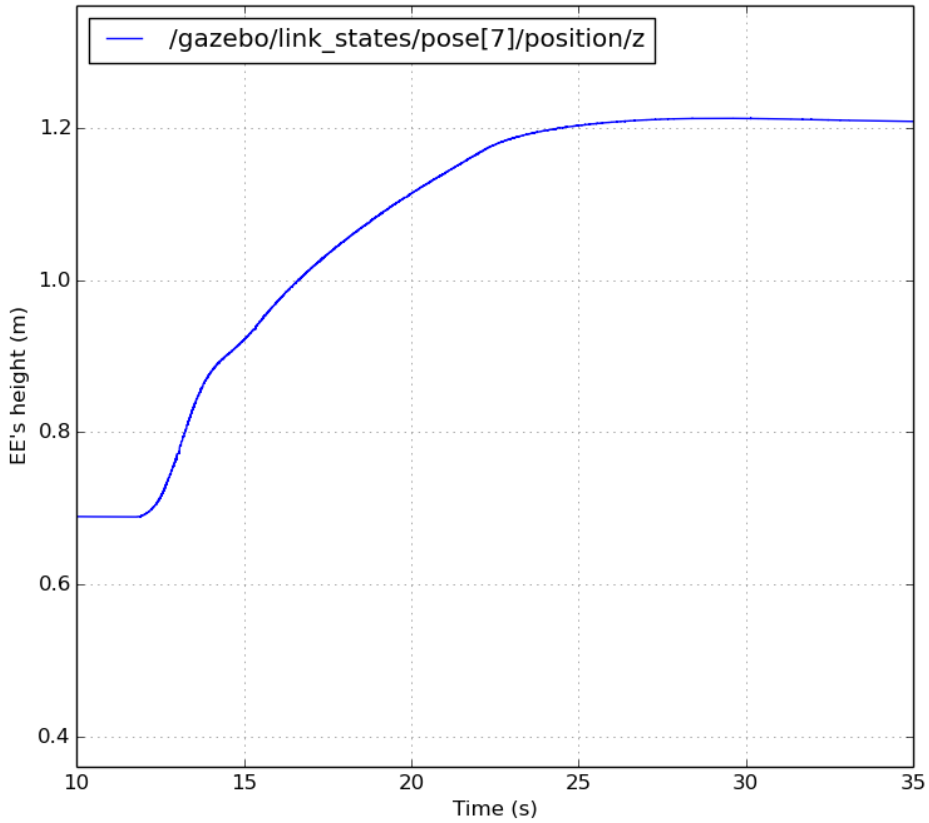

Figure 3.17. Simulation results #3.

After iterating over the first parameter the second gain (shoulder_lift_joint) underwent a similar process. Figure 3.18 shows two iterations were on the left the proportional gain was reduced to 600 and on the right to 900. The rise time of the forth test can be estimated as: $t_r = 5.5\ s$ while the second one results in $t_r = 5.7\ s$. Comparing the results with Figure 3.15. Simulation results #1. Figure 3.15 which holds the default controller values the same conclusion can be drawn.

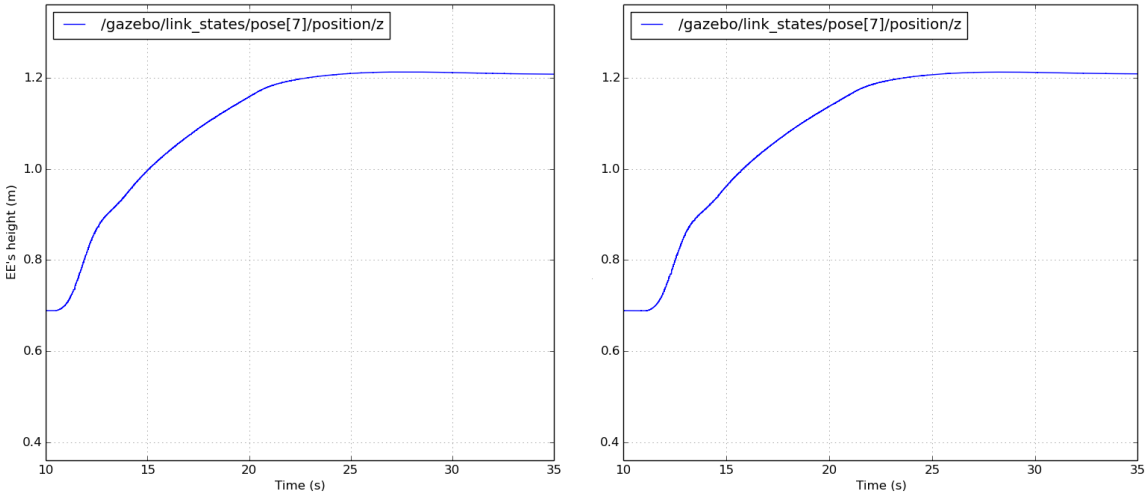

Figure 3.18. Simulation results #4 (left) & #5 (right).

A last set of two tests was performed using elbow_joint's proportional gain decreasing its value to 600 and 900 respectively. Figure 3.19 shows both tests being the one with a gain of 600 the left-most picture and 900 the rightmost. Their rise times were $t_r = 5.6\ s$ and $t_r = 5.9\ s$ which serves as final proof of the parameter's optimality.

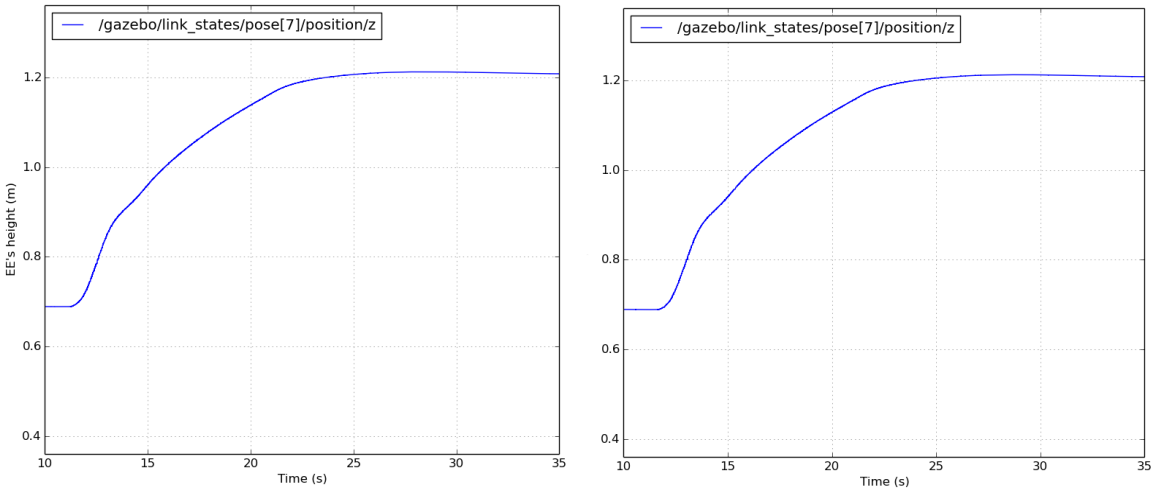

Figure 3.19. Simulation results #6 (left) & #7 (right).

3.2.5 Further tuning of PID controllers

In order to fully determine the stability of the PID controllers' parameters it was decided to use the tuning procedure presented by Kelly R. et al. in their book: Control of Robot Manipulators in Joint Space. The procedure ensures stability in the origin and was simplified by neglecting the vector of gravitational torques in their system of equations:

$$\boldsymbol{\lambda_{Max}\{K_i\} \geq \lambda_{Min}\{K_i\} > 0} \tag{10}$$

$$\boldsymbol{\lambda_{Max}\{K_p\} \geq \lambda_{Min}\{K_p\} > k_g} \tag{11}$$

$$\boldsymbol{\lambda_{Max}\{K_v\} \geq \lambda_{Min}\{K_v\} > \frac{\lambda_{Max}\{K_i\}}{\lambda_{Min}\{K_p\} - kg} \cdot \frac{\lambda^2_{Max}\{M\}}{\lambda_{Min}\{M\}}} \tag{12}$$

(Kelly et al., 2005, pp. 213-217.)

The system is simplified as previously mentioned by neglecting k_g. The related and necessary matrices are computed from Table 3.1 and UR10's STEP file obtained from Universal Robot's website. All of the matrices eigenvalues were obtained using Matlab's function: eig(). The result is shown below:

$$\begin{cases} \boldsymbol{\lambda_{Max}\{K_i\} = 500} \\ \boldsymbol{\lambda_{Min}\{K_i\} = 0} \end{cases} \rightarrow \boldsymbol{500 \geq 0 \geq 0} \tag{13}$$

$$\begin{cases} \boldsymbol{\lambda_{Max}\{K_p\} = 1500} \\ \boldsymbol{\lambda_{Min}\{K_p\} = 100} \end{cases} \rightarrow \boldsymbol{500 \geq 100 > kg = 0} \tag{14}$$

$$\begin{cases} \boldsymbol{\lambda_{Max}\{K_v\} = 100} \\ \boldsymbol{\lambda_{Min}\{K_v\} = 0} \\ \boldsymbol{\lambda_{Max}\{M\} = 3.84} \\ \boldsymbol{\lambda_{Min}\{M\} = -0.05} \end{cases} \rightarrow \boldsymbol{100 \geq 0 > \frac{500}{100} \cdot \frac{3.84^2}{-0.05}} \tag{15}$$

Since all three conditions are met (13-15) it can be concluded that the selected parameters for the PID controller ensure stability and are therefore optimally selected. A more thorough study could be performed by adding the vector of gravitational torques and a more detailed inertial matrix but it was considered out of the scope of this thesis.

4 CONCLUSION

This work has provided several results in the field of robotics due to the development of a custom maintenance robot. Although the robot is still undergoing changes and no physical entity exists yet as a whole much of the work has already been completed therefore once the components are assembled and the parts connected the remaining processes until completion will be minimal.

First of all this thesis has provided the necessary knowledge to set up a ROS network bridging together all related and necessary components in a clean and simple fashion. ROS has enabled a control network which mostly avoided low level programming through code reuse and package building. Most of the programming was done on parsers or interpreters which received commands from the joystick or some other media and through simple programming sent the required signal to an already existing part of the network.

Secondly stability information was acquired based on the tip-over stability analysis concluding that even in the worst case scenario over 900 N would need to be applied at the end effector's tip for the robot to lose one contact point with the ground. Considering the robot will not be subject to high accelerations neither linear nor due to the arms' movement it can be stated that the robot is stable from a tip-over perspective.

A third conclusion was drawn in the simulated environment which proved that ROS could handle the two pivotal parts of the robot: arms and driving system. After creating the whole URDF file of the robot and launching MoveIt a path planner was created which ensured a stable path finding algorithm. Such algorithm prevented self-collisions in real time thanks to the robot state publisher, which kept track and posted the current and past position of each joint and link in the robot.

Finally an attempt was made to optimize each arm's PID joint-controllers by iterating over the proportional gain of three of the joints. The results are summarized in Table 4.1 showing their optimal minimum at their default values (3.5 s). Although only one parameter (rise time) was taken into account for this test all other parameters worsened or remained the

same. It was concluded that changing the gains does not result in a desirable improvement of the arm's behavior while there remains a risk of unbalancing the integral clamping term which might result in a steady state error over time and might ruin the robot arms' accuracy. It can therefore be concluded that the controllers' gains are in their optimal state (as previously shown in Table 3.1) and no changes need to be done. Both arms act swiftly and accurately at least in the simulated environment.

Table 4.1.Simulation Results Summary..

	$K_p = 600$	$K_p = 900$	$K_p = 1500$
	$t_r\ (s)$		
Shoulder_pan_joint	3.5	4.7	5
Shoulder_lift_joint	5.5	5.7	3.5
Elbow_joint	5.6	5.9	3.5

It was proved that the already existing parameters of the whole system ensure stability at the origin and have therefore been selected through a tuning procedure. As a final note it should be stated that although the learning curve of such a unique solution (ROS) might result steep the obtained advantages after passing the curve outweighs the effort.

5 SUMMARY

This work has been written and carried out based on a new robot being developed at the Laboratory of Intelligent Machines in Lappeenranta University of Technology, Finland. The robot is being designed so that it can perform maintenance operations in environments dangerous or somewhat harmful to human beings. It will be controlled through tele operation by a distant driver sitting in a control room and using sensorial information being broadcasted from the robot.

To bridge every sensor and actuator together with the human operator it was decided to use an open source solution called: ROS. ROS relies on a package database of simple nodes which can be connected together to achieve complex structures. The network of nodes can be spread across different computers therefore it was the perfect solution for such a system.

The necessary nodes and packages have been introduced throughout section 2.2 so that the same node network can be reproduced without any further knowledge. In following sections the stability of the system was put to the test first as a physical dynamic system which was subject to tipping-over. It was concluded that the design was safe from such a phenomena after which a simulation model was developed. The model was further used for path planning and execution of end effectors to prove the full awareness of the algorithm preventing self-collisions whenever they were planned.

This work has concluded that the robot under development will have a stable behavior and that ROS can take care of most of its tasks without having to take care of low level programming.

LIST OF REFERENCES

Gazebo. 2014. [Gazebo webpage]. Updated November 4, 2015. [Referred 11.4.2015]. Available: http://gazebosim.org/

Grantham, D. W. 1995. [Chapter 9:] Spatial Hearing and Related Phenomena. In: Moore, B. C. J. (editor). Hearing. Academic Press, Inc. pp 297-345.

Kelly, R., Santibáñez V. & Loría, A. 2005. Control of Robot Manipulators in Joint Space. London: Springer-Verlag London Limited. 426 p.

Lewis, G. H. 1875. Problems of Life and Mind. Boston: Houghton, Osgood and company. 199 p.

Maxon Motors. 2013. [Online Document] EPOS2 70/10 Positioning Controller: Getting started. [Referred 2.11.2015]. Available: http://www.maxonmotor.com/medias/sys_master/8811457609758/375711-Getting-Started-En.pdf

Minxiu, K., Lining, S. & Yanbin, D. 2012. Control System Design for Heavy Duty Industrial Robot. The Industrial Robot vol. 39, no. 4 (2012), 365 p.

MoveIt. 2015. [MoveIt webpage]. [Referred 11.3.2015]. Available: http://moveit.ros.org/about/

Papadopoulos, E. G. & Rey, D. A. 1996. A New Measure of Tipover Stability Margin for Mobile Manipulators. Proc. IEEE Int. Conf. on Robotics and Automation Minneapolis, MN, April 1996.

Peppoloni, L., Brizzi, F., Avizzano, C. A. & Ruffaldi, E. 2015. Immersive ROS-integrated Framework for Robot Teleoperation. IEEE Symposium on 3D User Interfaces 2015, 23 - 24 March, Arles, France.

ROS. 2012. audio_common. [ROS Wiki webpage]. Updated September 7, 2012. [Referred 11.4.2015]. Available: http://wiki.ros.org/audio_common

ROS. 2013. Joint element. [ROS Wiki webpage]. Updated June 5, 2013. [Referred 11.3.2015]. Available: http://wiki.ros.org/urdf/XML/joint

ROS. 2014a. Concepts. [ROS Wiki webpage]. Updated June 15, 2014. [Referred 11.3.2015]. Available: http://wiki.ros.org/ROS/Concepts

ROS. 2014b. Link element. [ROS Wiki webpage]. Updated September 26, 2014. [Referred 11.3.2015]. Available: http://wiki.ros.org/urdf/XML/link

ROS. 2014c. URDF. [ROS Wiki webpage]. Updated December 10, 2014. [Referred 11.3.2015]. Available: http://wiki.ros.org/urdf

ROS. 2015a. Multiple Machines [ROS Wiki webpage]. Updated July 4, 2015. [Referred 3.11.2015] Available: http://wiki.ros.org/ROS/Tutorials/MultipleMachines

ROS. 2015b. Understanding Topics. [ROS Wiki webpage]. Updated July 21, 2015. [Referred 7.21.2015]. Available: http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS. 2015c. Universal Robot. [ROS Wiki webpage]. Updated June 12, 2015. [Referred 11.3.2015]. Available: http://wiki.ros.org/universal_robot

ROS. 2015d. About. [ROS webpage]. Updated July 21, 2015. [Referred 7.21.2015] Available: http://www.ros.org/about-ros/

ROSforLabview. 2015. [ROSforLabview webpage] [Referred 11.3.2015]. Available: https://sites.google.com/site/rosforlabview/

Scopus. Analyze Search 2015. [Scopus webpage]. Updated July 29, 2015. [Referred 7.29.2015]. Available: http://www-scopus-com.ezproxy.cc.lut.fi/term/analyzer.url?sid=80CE2480BA0EE57112B2C5D96D2AC9EE.

53bsOu7mi7A1NSY7fPJf1g%3a660&origin=resultslist&src=s&s=TITLE-ABS-KEY%28ROS%29&sort=plf-f&sdt=cl&sot=b&sl=18&count=2831&analyzeResults=Analyze+results&cluster=scosubjabbr%2c%22BIOC%22%2cf%2c%22MEDI%22%2cf%2c%22PHAR%22%2cf%2c%22AGRI%22%2cf%2c%22CHEM%22%2cf%2bscosubjabbr%2c%22NEUR%22%2cf%2c%22IMMU%22%2cf%2c%22ENVI%22%2cf%2bscosubjabbr%2c%22EART%22%2cf%2c%22VETE%22%2cf%2c%22DENT%22%2cf&txGid=80CE2480BA0EE57112B2C5D96D2AC9EE.53bsOu7mi7A1NSY7fPJf1g%3a71

Sucan, I. 2015. MoveGroup Class Reference. [ROS Docs webpage] Updated November, 2015. [Referred 11.3.2015] Available: http://docs.ros.org/jade/api/moveit_ros_planning_interface/html/classmoveit_1_1planning__interface_1_1MoveGroup.html

Teymourzadeh, R., Mahal, R.N., Shen, N.K. & Chan, K.W. 2013. Adaptive intelligent spider robot. Proceedings - 2013 IEEE Conference on Systems, Process and Control, ICSPC 2013.

Tlale, N. & De Villiers, M. 2008. Kinematics and Dynamics Modelling of a Mecanum Wheeled Mobile Platform. Available: http://researchspace.csir.co.za/dspace/bitstream/10204/2771/1/Tlale2_2008.pdf

Trossen Robotics, 2015. [Trossen Webpage]. Updated September 14, 2015. [Referred 9.14.2015]. Available: http://www.trossenrobotics.com/p/phantomX-robot-turret.aspx

Visioli, A. 2006. Practical PID Control. Publisher: Springer. 314 p.

Zimmerman, K., Zeidis, I. & Abdelrahman, M. 2014. Duynamics of Mechanical Systems with Mecanum Wheels. In: Awrejeewicz, J. (editor). Applied Non-Linear Dynamical Systems. Springer. pp. 269-279.

APPENDIX 1

Simulation’s Launch File.

```
<launch>
   <include file="$(find centaur_gazebo)/launch/centaur.launch"/>
  <arg name="sim" default="true" />
  <arg name="debug" default="false" />

  <!-- Remap follow_joint_trajectory -->
  <remap if="$(arg sim)" from="/follow_joint_trajectory"
to="/right_arm_controller/follow_joint_trajectory"/>
  <remap if="$(arg sim)" from="/follow_joint_trajectory"
to="/left_arm_controller/follow_joint_trajectory"/>

  <!-- Launch moveit -->
  <include file="$(find centaur_1_moveit_config)/launch/move_group.launch">
    <arg name="debug" default="$(arg debug)" />
  </include>

   <include file="$(find centaur_1_moveit_config)/launch/moveit_rviz.launch">
      <arg name="config" default="true"/>
   </include>
<!--node name="centaur_planner" pkg="centaur_planner" type="planner" respawn="true"
/-->
</launch>
```

APPENDIX 2

Neck ArbotiX's code.

```
#if (ARDUINO >= 100)
 #include <Arduino.h>
#else
 #include <WProgram.h>
#endif
#include <Servo.h>
#include <ros.h>
#include <std_msgs/UInt16.h>
#include <ax12.h>
#include <sensor_msgs/Joy.h>

ros::NodeHandle  nh;
void servo_cb( const sensor_msgs::Joy& cmd_msg){
  SetPosition(1,2048+cmd_msg.axes[1]*2048);
  SetPosition(2,2048+cmd_msg.axes[4]*1024);
  digitalWrite(0, HIGH-digitalRead(0));  //toggle led
}
ros::Subscriber<sensor_msgs::Joy> sub("joy", servo_cb);
void setup(){
  pinMode(0, OUTPUT);
  nh.initNode();
  nh.subscribe(sub);
  SetPosition(1,2048);
  SetPosition(2,2048);
}
void loop(){
  nh.spinOnce();
  delay(1);
}
```