ARPA: Autonomous Robotic Pointer Arm

Helgerud, Erlend Håland, André

April 4, 2018

Abstract

This report is written as part of a project in the course DFMS3200 - Simulation and Modelling at the University College of South-Eastern Norway. The background for the project is the idea of automation in industrial production lines in order to minimize costs and increase productivity. The solution proposed is an autonomous 5DOF robotic manipulator that perform actions based on a live video feed. The main focus of the project is to serve as a introductory case-study in robotic manipulation and control, and therein the usage of selected tools. The project is split in two, virtual simulation and physical representation. For controlling both the virtual and physical representation, the project utilizes ROS Kinetic (Robotic Operating System) in Ubuntu 16.04 with the majority of code written in C++. The physical robot is powered by 12 AA batteries together with two Arduino MEGA2560's. The vision system is based in Python 2.6 and OpenCV2. In order to provide the virtual representation the project incorporates Gazebo together with needed models written in URDF/Xacro and configurations written in YAML. Kinematics are handled in Matlab's Robotics System Toolbox. As a final demonstration, the separate components of the project will be run with a 1:1 relationship between the virtual and physical representations, integrating all components of the project.

Contents

List of Figures

List of Tables

1 Introduction

As a result of the high average of salaries in Norway a lot of the processing of food is outsourced to countries with a lower average salary [1]. An example is the outsourcing of processing of Norwegian salmon to Poland, which has a lower hourly rate of work by 13.5% compared to Norway. A direct consequence of the difference is that companies like Marine Harvest export roughly processed salmon to Poland, and import fine processed salmon back to Norway [2]. To be able to preserve the processing of fish in Norway, there is a demand for a system that can remove defects such as blood and/or melanin defects from the surface of fish fillets.

Using the previous paragraph as the background, the motivation for this project is to conduct case-study in robotic manipulation and control in order to be familiarized with possible tools and technologies used in robotics.

The project itself has served as an introduction to robot technology, thus, a big part of the project has been the learning of the different technologies which will be discussed in detail later in the report.

1.1 Goals

The following goals were set for the project:

- Gain knowledge in Robotics Operating System
- Develop a fully functioning prototype
  - 1:1 relationship between physical and virtual model
  - Actions based on object-detection
  - Develop own kinematic model

1.2 Contributions

The project aims to contribute an environment consisting of a physical 5DOF robotic manipulator with a 1:1 relationship to a virtual simulation which can be used for rapid prototyping and testing. The system has incorporated a detection system to automate the process of finding markers. In addition, the delivered report and GitHub repository should serve as a contribution to student inheriting the project at a later stage.

The physical robot is powered by two Arduino Mega2560's and 12 AA batteries. One of the Arduinos functions as the controller for the physical robot, while the other one serves solely as a power supply to a servo motor.

1.3 Tools

The tools used in ARPA, both in terms of hardware and software are presented in the following sections.

1.3.1 Hardware

Table 1 shows a list of the hardware components used in ARPA. The robotic manipulator is not listed, nor are bare necessities such as wiring etc.

Hardware		
Component	**Quantity**	**Usage**
Hitec HS-475HB Servo	1	4th joint
Sanwa Hyper ERG-VZ Servo	2	2nd and 3rd joint
Savox Digital SC0252MG Servo	1	1st joint
Parallax Standard Servo	1	5th joint
Creative Labs Webcamera	1	Detection system
Arduino MEGA2560	2	Control and power
4x AA Battery Pack	3	Power

Table 1: Hardware used in ARPA

1.3.2 Software dependencies

Table 2 shows a list of the software dependencies for running ARPA.

Software	
Software	**Usage**
Ubuntu 16.04 LTS	Operating System
ROS Kinetic	Framework for robot software
Gazebo	Simulation environment
rosserial_arduino	Include Arduino into ROS node network
MATLAB: Robotics System Toolbox	MATLAB add-on providing different robotic software
OpenCV2	Computer vision library used with Python

Table 2: Software dependencies

The choice of using ROS was made quite quickly, both due to a good pitch from the supervisor, but also by discussing the actual value of learning it. Since both participants of the project are conducting the bachelor's thesis in parallel with this project, the learning outcome of ROS was evaluated to be highly relevant for our other projects aswell.

When choosing the simulation environment for the project, we focused mainly on the return of investment in terms of learning the simulation environment. In addition to this it was important that the chosen environment had support for Arduino. A last factor was that we needed a software which was free, or atleast had free student licenses. In general, we decided that we wanted to use ROS (Robotic Operating System) as the robotic framework for our system. As a result, the simulation environment to be used had to support ROS. *Figure 1* shows a comparison chart of the environments discussed in the group.

	OS	Language	Tutorials	License	ROS connectivity	Arduino
V-REP	Windows, Linux, MacOS	C/C++, Python, Matlab	Yes	Commercial - free educational version	Plugin exists	Yes
Gazebo	Li nux	C/C++, Python	Yes	Open source	Maintained alongside ROS	Yes
Webots	Windows, Linux, MacOS	C/C++, Python, Matlab	Yes	Commercial - educational costs money	Has ROS interface	Yes
Player/Stage	**Player:** Windows, Linux, MacOS. **Stage:** Linux, MacOS	**Player:** Any that supports TCP sockets. **Stage:** C/C++, Python	Yes	Open source	Has ROS interface	Yes

Figure 1: Comparison chart for simulation environments

In the process of choosing the simulation environment, we mainly discussed V-Rep and Gazebo. Since we established the fact that we wanted to use ROS, the most natural choice for the environment became Gazebo, since this is maintained with ROS. We also considered the return of investment in terms of future usage. The two factors which kept us hesitant when deciding for ROS was that it is Linux based (none of the group members had computers set up with Linux), and the fact that we were informed by the teacher in the subject that the learning curve is steep. The challenge with no Linux-based computers was solved by installing Linux on two old computers.

1.4 Content

In addition to the introduction, this report consists of **2: Related works**, **3: System description**, **4: Simulations and experiments** and **5: Discussion**.

2 Related works

Robotics is a wide field, and many projects have been conducted. This ection will compare some related work to this project.

2.1 Robot arm control with Arduino

The work in this report is conducted by Abdellatif Baba. Need for help systems in everyday life, and the evolution of autonomous systems is the background for the project. To solve this, Baba is looking to design a robot arm which takes a piece of material and moves it to a desired position. The movement should be recorded, and the arm should continue to do the action until the user tells it to stop [3]. Table 3 shows a comparison of ARPA and Robot arm control with Arduino (RACWA)

Comparison	
ARPA	**RACWA**
Background in automation	Background in automation
Uses Arduino Mega2560	Uses Arduino Nano
Uses five servo motors	Uses five servo motors
Wired communication between PC and Arduino	Communication through Bluetooth on Arduino with Android application
Uses ROS	Does not use ROS
Model visualized in simulation environment	No visual model

Table 3: Comparison of ARPA and RACWA

2.2 Anthropomorphic Robotic Manipulator

The work in this report is conducted by Jeffry Walker, Joshua Beverly, Christopher Cook and Migueal de Rojas. The goal of the project was to design a 5 DOF Anthropomorphic Geometry Manipulator (APGM) with a gripper as end-effector. The end-effector should be able to lift a cube with lengths of 5cm and mass of 0.34kg and place it between two predetermined spatial coordinates [4]. Table 4 lists some of the differences and similarities between ARPA and Anthropomorphic Robotic Manipulator (ARM).

Comparison	
ARPA	**ARM**
Five DOF	Five DOF
A simple poker as end-effector	Parallel opening gripper as end-effector
Moves to camera detected marker	Moves to predetermined points and picks up object and moves it to predetermined spot
Uses five servo motors	Uses five DC motors with optical encoders used to PID regulate
Uses Arduino Mega	Uses Arduino Mega

Table 4: Comparison of ARPA and ARM

3 System description

3.1 Architecture

The idea of ARPA is to have a 1:1 relationship between a physical robotic arm, and a model in Gazebo. This means that the model in Gazebo should contain (as good as) the same measurements as the physical robot. They will also receive the same control signals leading to synchronized movement. Figure 2 show this relationship between the real world and the simulated world.

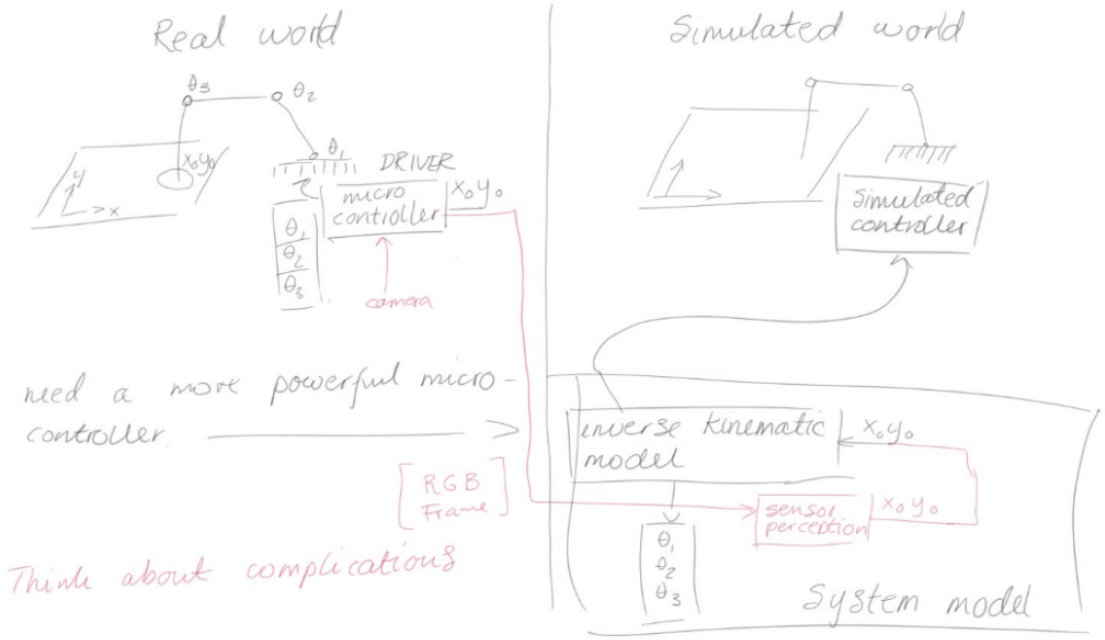


Figure 2: System architecture

ARPA is divided into nodes. These nodes each has its own tasks and communicate with each other on different topics to exchange information. Figure 3 shows the nodes of ARPA, and the topics they communicate on. A circle represents a node, whilst a rectangle represents a topic. The controller node is the center of all actions. It handles the interface between the camera, MATLAB, the physical robotic arm and the model in gazebo. The camera node sends a Cartesian point in space to the controller. The MATLAB node handles inverse kinematics. It receives Cartesian coordinates from the controller, and sends back five joint angles. The 5DOF Robot node controls the physical robotic arm by writing received angles to the servo motors. The Gazebo node listens to the same topics as 5DOF Robot, and will simulate a robotic model with a 1:1 relationship with the physical arm.

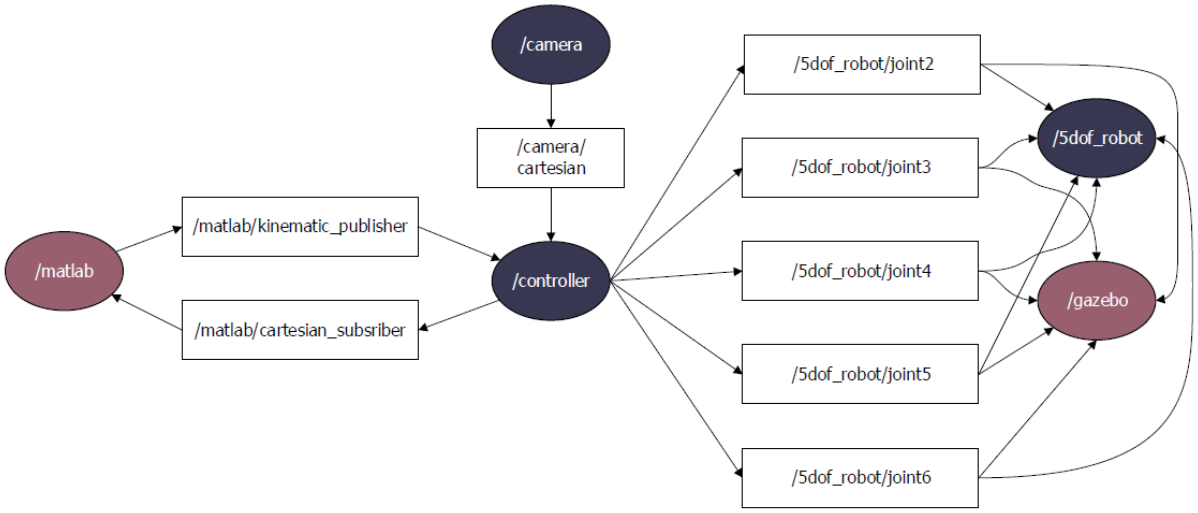


Figure 3: Node graph

As described in *4.5 Inverse kinematics*, we had some challenges with finding the inverse kinematics and have to depend on the Robotics System Toolbox from MATLAB. When running MATLAB as a ROS node, we discovered that it ran quite slowly together with Gazebo. For this reason, we decided to run the ROS network over two computers to separate computing power. This is done by setting a couple of environmental variables in every terminal that uses ROS. Two shell scripts were made for this, and can be sourced to achieve the distributed ROS network. The different colored nodes in figure 3 represents which computer the node runs on.

The nodes previously described is made by ROS packages. Figure 4 shows the packages used in ARPA. The following subsections will describe each package, including why and how they are implemented.

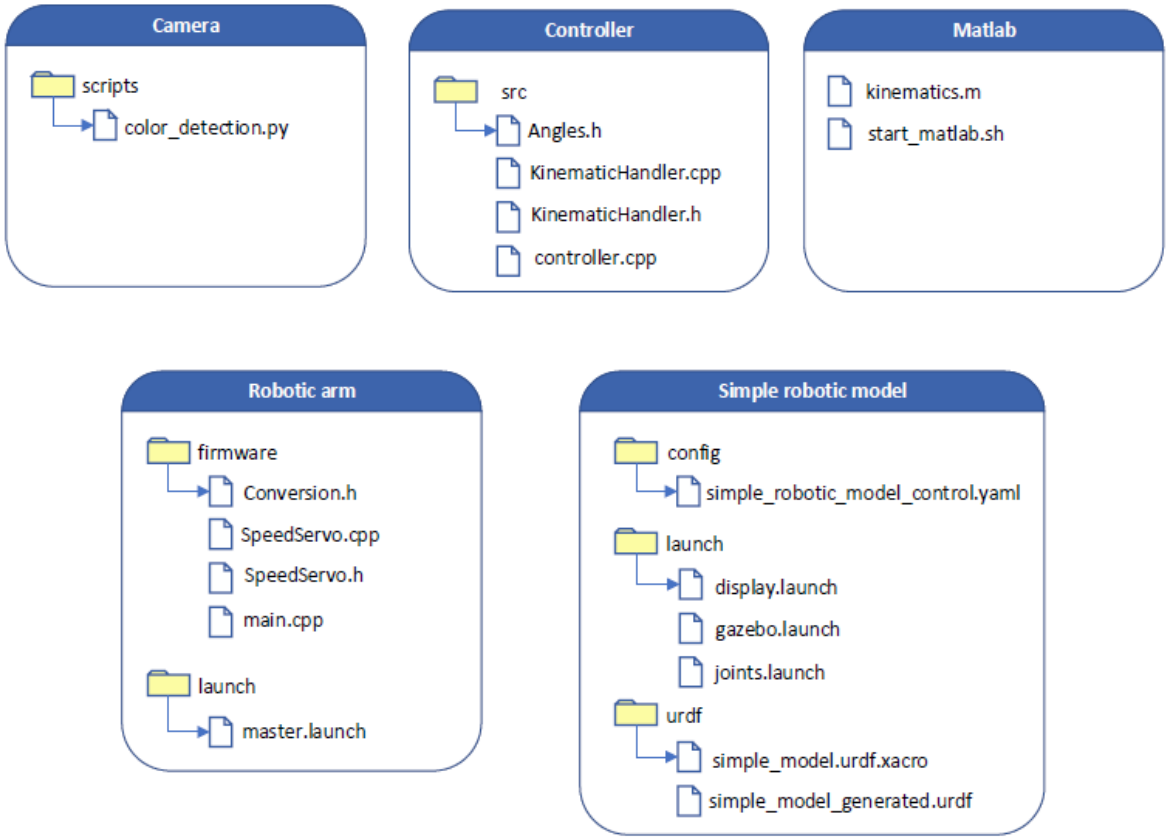


Figure 4: Packages

3.1.1 Camera

The camera package contains the python-script that is responsible for handling object detection in ARPA. By placing red markers in the physical coordinate system of ARPA, the user will be able to track the markers with the live video feed from OpenCV. The script will after calibration continuously map received coordinates from the OpenCV coordinate frame to coordinates corresponding to the physical coordinate frame on the rig. The calibration and usage during runtime will be elaborated in finer detail in *3.4.1 Calibration of camera*.

The color-detection is done by applying a red mask in the HSV-colorspace on each frame from the live video. By finding the contours of the image the marker can be detected and the script further finds the center of the detected point. The script is incorporated in the ROS environment as a node that publishes the mapped cartesian x and y coordinates of a desired point. The script can be extended to detect more colors by adding additional

3.1.2 Controller

The controller package handles the interface between the other nodes. First of all, it receives all the coordinates from the camera node, but will only publish them to MATLAB if the *ready.sh* script has been run. Optionally, a message containing a "1" (boolean true) can be published on the*/camera_ready* topic. After the controller has published the coordinates to MATLAB, it will receive a response in the form of five joint angles. These joint angles are further sent to their respective topics. Immediately after the angles are published, the controller sleeps for five seconds. During these five seconds, the robotic arm will have time to go to the given coordinates. After sleeping, the controller will start a function that sets the arm to a upright position, hereby referred to as the start position of the arm. It does so by rotating the top servos the opposite direction of the bottom servos, before it rotates the bottom servos. This is done because the torque of the bottom servos are to big to directly set all joints to start position. After approximately using three seconds to set the arm to start position, the controller is ready to get a new ready signal from the user.

3.1.3 Matlab

The matlab package is responsible for solving the inverse kinematics for ARPA. The matlab script contained in kinematics.m is integrated in the ROS environment through the Robotics System Toolbox, allowing Matlab to act as a standalone node in the graph. The script subscribes to the cartesian_subscriber which receives messages from the camera. When messages are received, the callback is initialized which derives the inverse kinematics with a numerical approach.

The inverse kinematics solver is based on a URDF model of the robot, which means it can return inverse kinematics for any given 5DOF robotic manipulator. The script publishes the derived values of θ_n in the form of a Float64MultiArray to the kinematic_publisher, which in turn is subscribed to by the controller.

3.1.4 Robotic arm

The robotic arm package contains the software running on the Arduino to control the servo motors. The main task of this node is to receive joint angles from the controller, and set the servo motors to the correct position. To control the servo motors, the native servo library provided by Arduino is used.

Writing a joint angle directly to the servo motors makes the servos rotate at maximum speed. To slow down the angular velocity of the motors, the robotic arm package implements a class which slowly increments the write-value sent to the servos. This speed of the servos can then be changed by adjusting how often the write value should be incremented.

This package also contains the launch file running the controller- and camera node in addition to the robotic arm node. Refer to *3.4 Starting program* for how to use the launch file.

3.1.5 Simple robotic model

The simple robotic model package contains the URDF/Xacro used to model the virtual representation of the robot. It contains one Xacro file, which is programmed, and the generated URDF file that is used in Gazebo. The launch files are located in the launch directory, where joints.launch is the master launch file. This file includes the generated URDF model aswell as it includes the gazebo.launch file. The joints.launch will also launch the MATLAB node. The YAML file simple_robotic_model.control in the config directory contains the configurations for the joint_state_controller, which handles position control of the joints of ARPA.

3.2 Information flow

The flow of information in ARPA starts when the user sends a ready signal to the controller node. This signals that the controller will accept a Cartesian coordinate from the camera. This coordinate further sent to MATLAB for processing. MATLAB returns five joints angles to the controller. These are the angles used to control both the physical arm and the Gazebo model. Hence, the controller publishes them to their respective topics, where they are picked up by both the Gazebo node and the 5DOF Robot node. After the control node has sent the joint angles, it waits for 5 seconds before it sends a new set of joint angles used to set the physical and simulated arm back to starting position. This process can be repeated by sending a new ready signal. Figure 5 shows the flow of information in the form of a UML Sequence diagram.

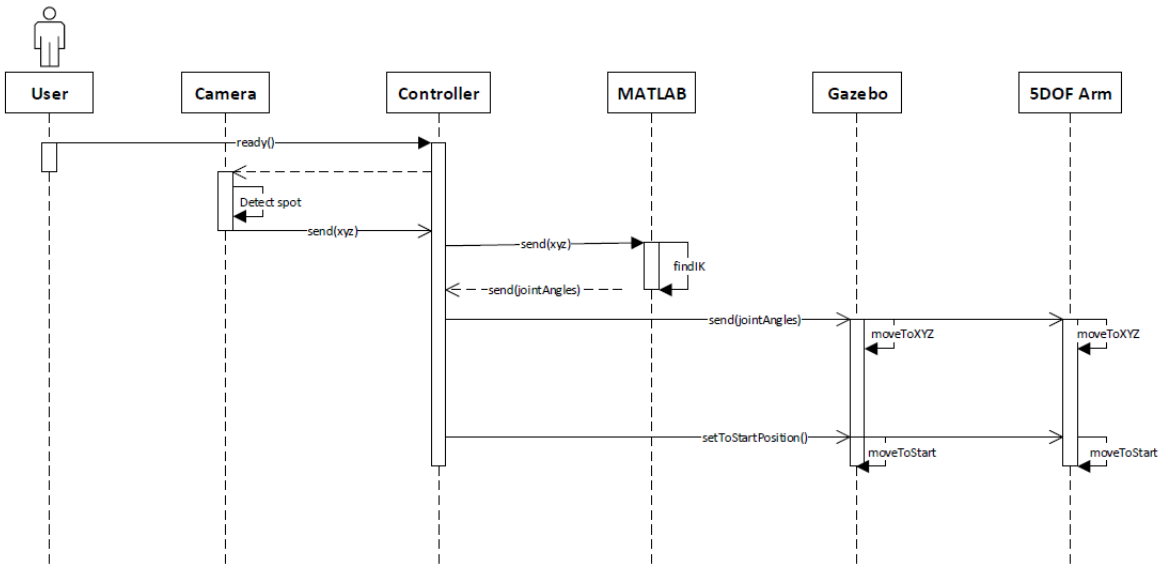


Figure 5: UML Sequence diagram

3.3 Use case

The different actors involved in ARPA and their main use cases are shown in the form a a UML use case diagram in figure 6.

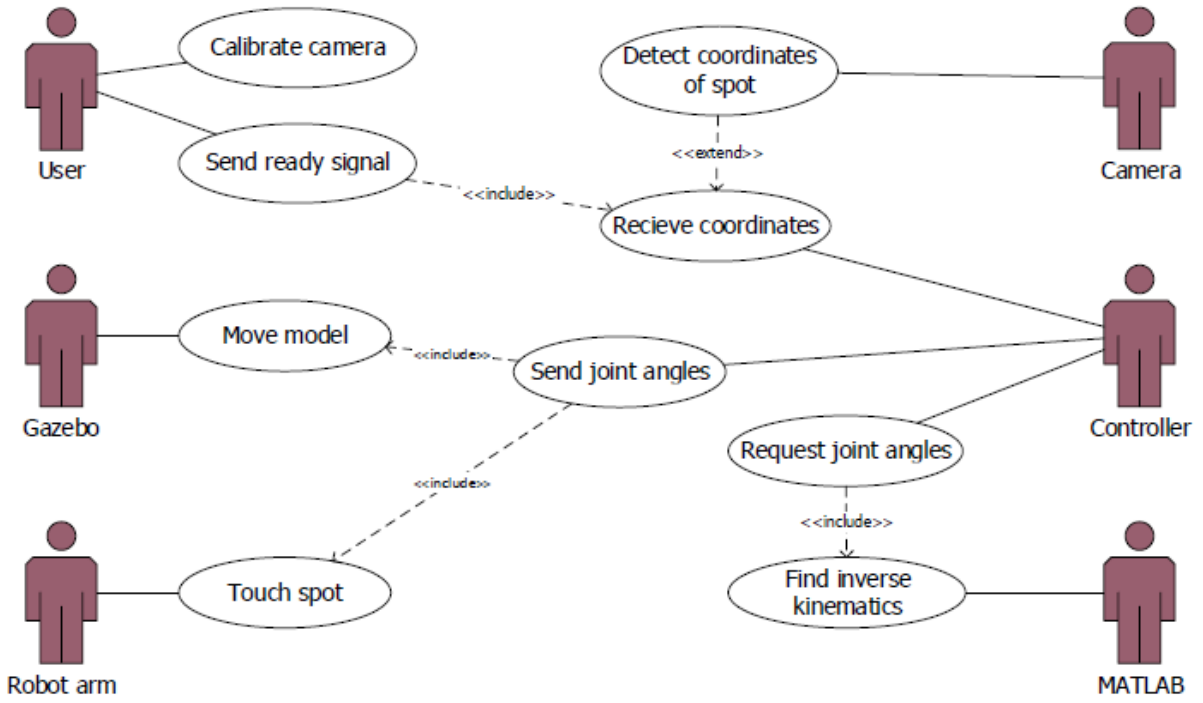


Figure 6: UML Use case

3.4 Starting program

When running ARPA over two computers, one of them must be chosen to run as the ROS master. The selected master computer must source the master.sh script in every terminal used for running ROS commands. Next, the slave.sh script must be modified to contain the IP-address of the master computer. The address can be read from the output of the master.sh script. When the slave script is modified to contain the correct address, it must be sourced in all terminals on the slave computer that runs any ROS commands.

Next up is to launch all ROS nodes. First of all, the setup.bash script in the devel directory of the catkin workspace must be sourced. When this is done on the master computer, the Arduino controlling the servos of the robotic arm must be connected to the computer. When it is connected, the command *roslaunch robotic_arm master.launch* can be run. Note that the launch file is configured to setup the ROS node on /dev/ttyACM0. This means that either the Arduino running the code must be connected before the Arduino used for power supply (see *3.6.2 Power Supply*), or the launch file must be modified to contain the correct device parameter. The roslaunch command starts the camera, controller and the 5DOF robot node.

3.4.1 Calibration of camera

The detection system used in ARPA is based in Python and the OpenCV library, and can be used together with an internal or external camera. The python script is in the beginning made so that a user has to go through a calibration phase to ensure mapping from the OpenCV coordinate frame to the coordinate frame of the physical rig.

Before running *roslaunch robotic_arm master.launch* there should be a red marker placed in ARPA's origo. The x-axis on the physical rig should also be parallel with the bottom border of the OpenCV window frame. The camera will during the first five seconds self-calibrate for lighting and focus, before masking the picture to find the red marker representing origo. After capturing 50 frames there is a five second pause in which the user will have to move the marker from origo to x-max, after an additional 50 frames the same will be repeated for y. When the script has defined all needed points of the coordinate frame, the values from the OpenCV frame is mapped to the lengths of the physical coordinate frame defined in the script. This method ensures that the same calibration method can be used independent of the cameras height above the rig, as long as origo, x-max and y-max is in the camera-frame.

As with the master computer, the slave must also source the setup.bash script in the devel directory of the catkin workspace. When this is done, the command *roslaunch simple_robotic_model joints.launch* can be run. This starts both the Gazebo and MATLAB node.

3.5 User interface

The user interface of ARPA is split into two: one on the master computer and one on the slave computer. Figure 7 shows the user interface of the master. The window to the left is a live feed of the camera. It shows the user the detection of a red dot, i.e. where ARPA will move when given a ready signal. The window to the right allows the user to send this ready signal. When the ready.sh script is executed, the controller node will accept the next incoming coordinates and execute all necessary communications and actions for both the physical arm and the Gazebo model to move.

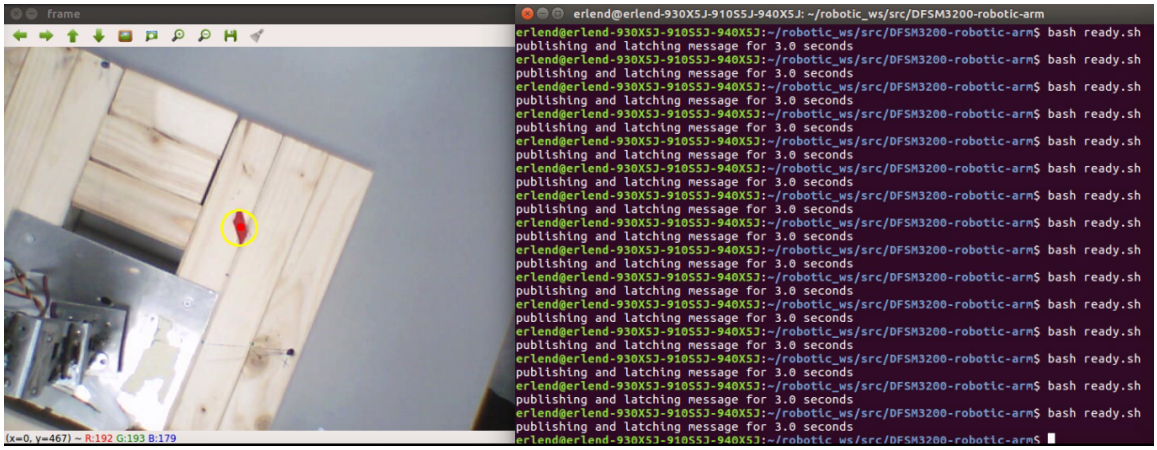


Figure 7: User interface: Master

Figure 8 shows the user interface of the slave. Gazebo is launched as described in *3.4 Starting program*, but the rqt window on the left is not included as a part of the launch file. The user can choose to start rqt from a new terming through the *rqt* command. The preferred setup for this project was to have rqt run the console plugin to show all the ROS messages in a GUI.

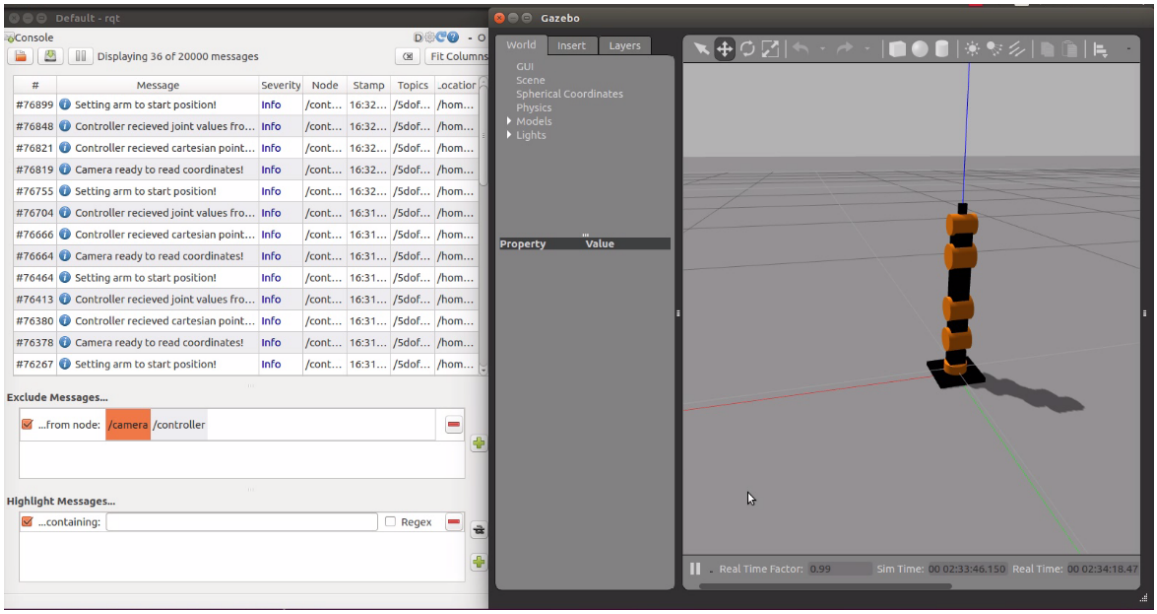


Figure 8: User interface: Slave

3.6 Hardware

During the project we had to make some adjustments to parts of the hardware related to the robotic arm. This section will go explain which adjustments has been made, and why.

3.6.1 Rig

When we were handed the robotic arm at the beginning of the course, we did some testing of it. First of all, we discovered that it both had some loose screws, and were also missing some. This lead us to spending some time on making the arm more rigid and secure. Figure 9 shows this stage of the project.

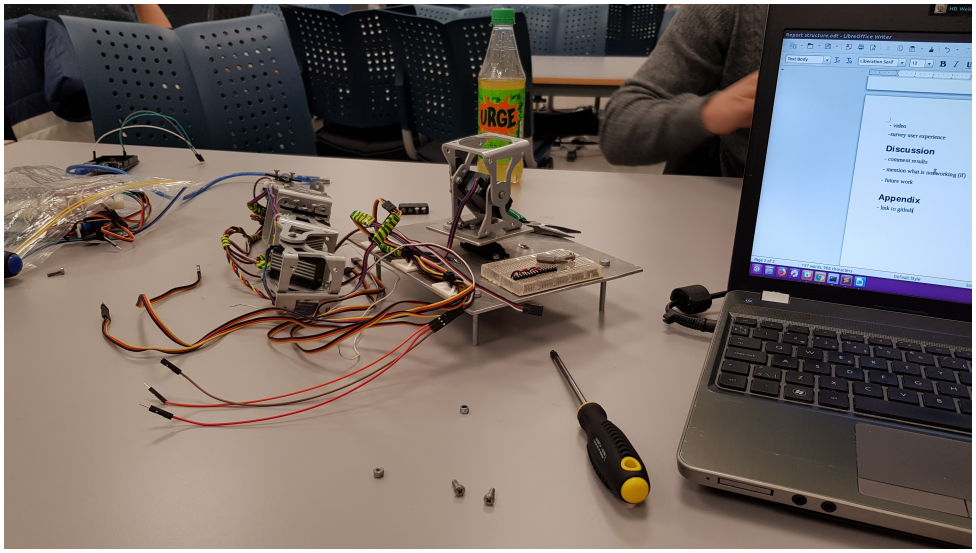
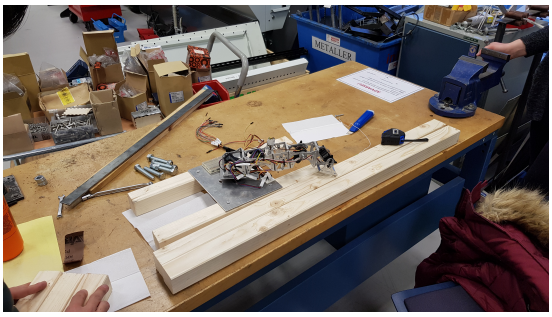
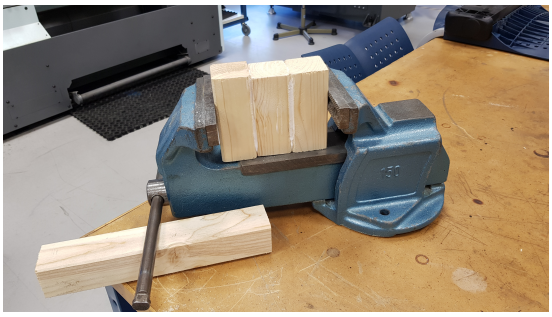


Figure 9: Early stage of the hardware

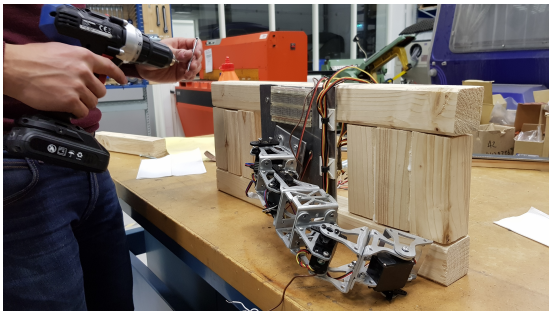
During the first tests of the robotic arm, we discovered that the torque would make the robot fall over. We decided to make a more stable mechanical rig, by attaching the robotic arm to a wooden platform. Figure 10 shows the process of making the wooden platform.



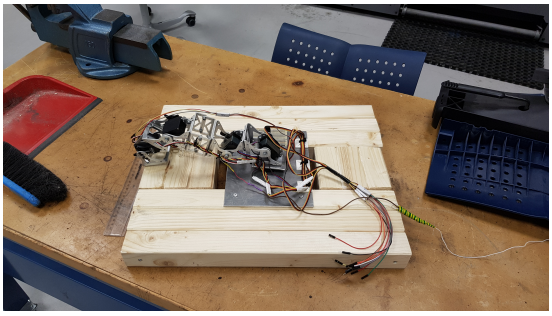
(a) Planning



(b) Adjusting



(c) Assembly



(d) Finished

Figure 10: Process of making the wooden rig

3.6.2 Power supply

The servo motors are running nominally at 6V. As the arm consists of five motors, we could not power them all from the Arduino. We decided to buy battery holders to run 4 AA batteries in series. Since an AA battery is 1.5V, running four of them in series adds up to 6V.

Due to the bigger torque applied to the bottom motors, we decided to supply the bottom three motors with 6V each, having them operate at nominal voltage level. The upper two motors of the arm are powered by one Arduino each, meaning they get 5V. Figure 11 shows the power supply used in ARPA. Note that the battery on the right holds six more batteries under the ones showed.

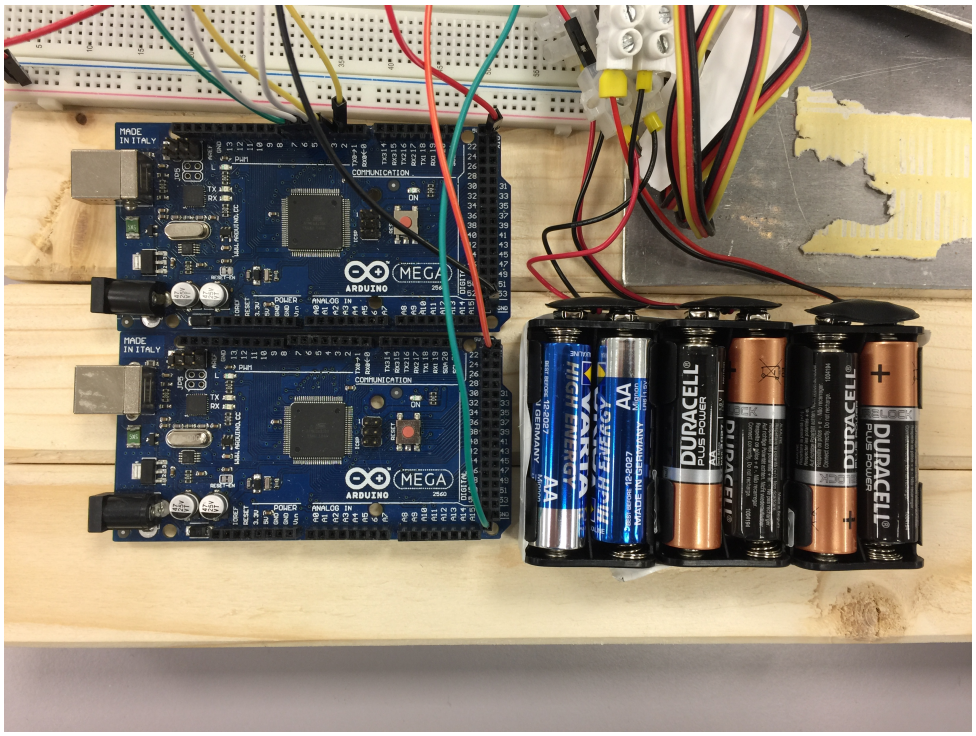


Figure 11: Power supply setup

3.6.3 Arduino wiring

Figure 12 shows the wiring of Arduino to supply power to the motors, as well as controlling them with PWM signals.

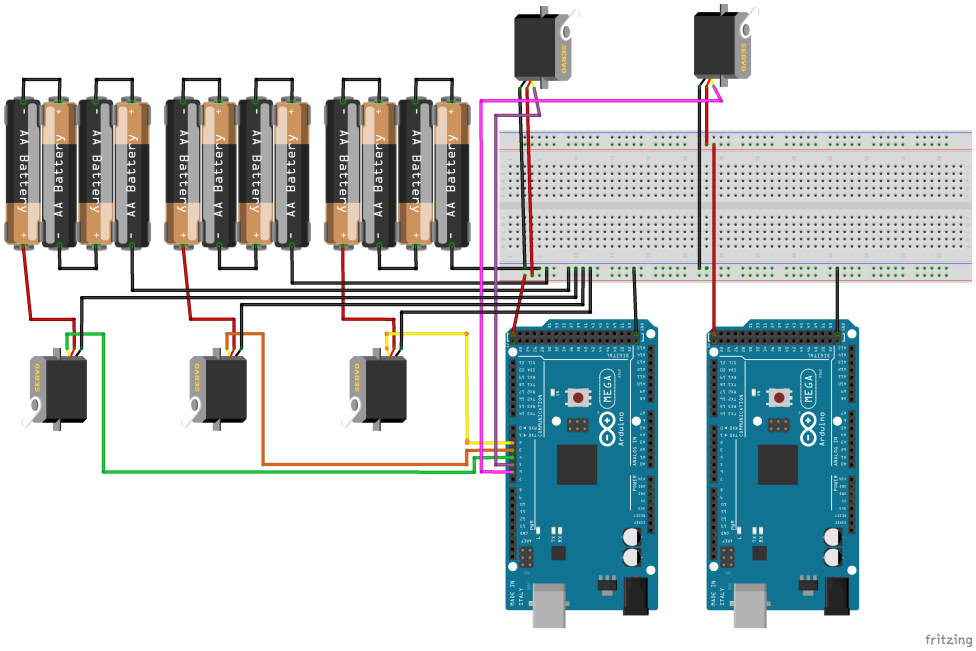


Figure 12: Arduino wiring

4 Simulation and experiments

Since the project has served as a introduction to many new tools, we have had several experiment along the way to learn how to use these tools. This section will go through the simulations and experiments we have conducted during the course of the project.

4.1 Arduino knob

The first experiment was to learn how to control the servo motors. We used a couple of approaches, one being to control the servo motors by the output of a potentiometer. It was a quick process, which allowed us to move onto bigger challenges.

4.2 ROS Turtlesim

The next stage in the project was to learn the ROS ecosystem. ROS has quite a steep learning curve, and much time was spent just getting a basic understanding of all the fundamentals. One specific experiment was the turtlesim tutorial found in [5]. This tutorial gave a good insight into the ROS graph concept, i.e. the separation of functionality into nodes.

4.3 Arduino subscriber

After getting a grip of the basics in ROS, we went on to integrating the Arduino into the ROS node network. The goal of this experiment was to setup a subscriber on the Arduino which receives a joint angle that it writes to a servo motor. This experiment was successful as we managed to rotate the servo by publishing a message from the terminal on the computer.

4.4 Initial modeling

The next experiment in the project was to get a simple robotic model visualized in Gazebo. Through guidance in [7], we were able to get a simple representation of the robotic arm with the correct amount of joints and links, but without precise D-H parameters. We managed to make this model subscribe to the same topics subscribed to by the Arduino so the joints in the simulated world received the same messages as the servo motors on the physical robot. Figure 13 shows the initial model of the robotic arm.

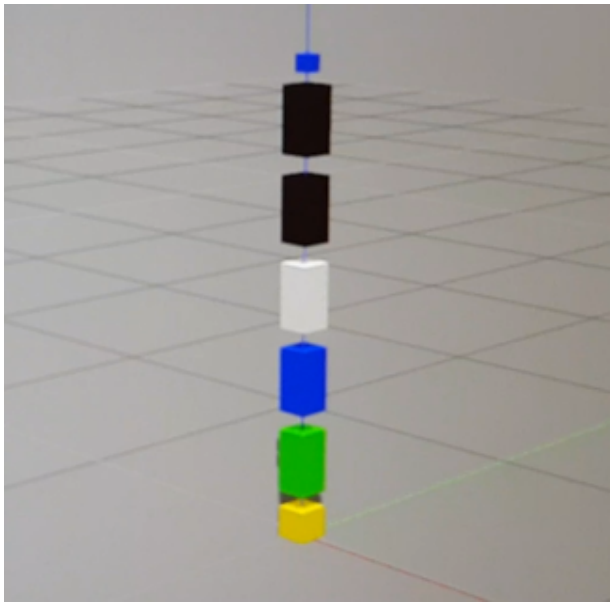


Figure 13: Initial model

4.5 Inverse kinematics

To convert Cartesian coordinates to joint space, we needed to find the inverse kinematics of the robotic arm. We used a symbolic approach to first derive the forward equation of the geometric representation shown in figure 14 follow by the solving of the inverse kinematics. The resulting equations for θ_1, θ_2, θ_3, θ_4, θ_5 are shown below. The mathematics in its entirety is appended as appendix *A Complete kinematic solution.*

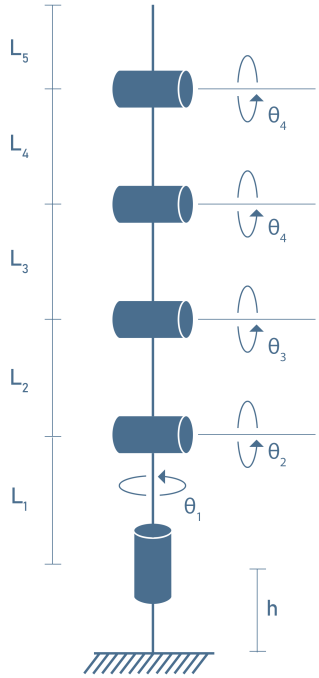


Figure 14: Geometric representation

$$\theta_1 = 2\text{atan}\left(\frac{p_x \pm \sqrt{-d_2^2 + p_x^2 + p_y^2}}{d_2 - p_y}\right).$$

$$\theta_2 = 2\,\text{atan}\left(\frac{a_z \pm \sqrt{a_x^2 c_{\theta_1}^2 + 2a_x a_y c_{\theta_1} s_{\theta_1} + a_y^2 s_{\theta_1}^2 + a_z^2}}{a_x c_{\theta_1} + a_y s_{\theta_1}}\right)$$

$$\theta_3 = \begin{cases} -2\,\text{atan}\left(\frac{\sqrt{a_x^2 c_{\theta_1}^2 + s_{2\theta_1} a_x a_y - a_y^2 c_{\theta_1}^2 + a_y^2 + a_z^2} - a_z c_{\theta_2} + a_x c_{\theta_1} s_{\theta_2} + a_y s_{\theta_1} s_{\theta_2}}{a_z s_{\theta_2} + a_x c_{\theta_1} c_{\theta_2} + a_y c_{\theta_2} s_{\theta_1}}\right) \\ 2\,\text{atan}\left(\frac{\sqrt{a_x^2 c_{\theta_1}^2 + s_{2\theta_1} a_x a_y - a_y^2 c_{\theta_1}^2 + a_y^2 + a_z^2} + a_z c_{\theta_2} - a_x c_{\theta_1} s_{\theta_2} - a_y s_{\theta_1} s_{\theta_2}}{a_z s_{\theta_2} + a_x c_{\theta_1} c_{\theta_2} + a_y c_{\theta_2} s_{\theta_1}}\right) \end{cases}$$

$$\theta_4 = \begin{cases} -\theta_2 - \theta_3 - \text{asin}\left(\frac{d_1 - p_z + a_6 n_z + a_4 s_{\theta_{23}} + a_3 s_{\theta_2}}{a_5}\right) \\ \pi - \theta_2 - \theta_3 - \text{asin}\left(\frac{d_1 - p_z + a_6 n_z + a_4 s_{\theta_{23}} + a_3 s_{\theta_2}}{a_5}\right). \end{cases}$$

$$\theta_5 = \begin{cases} -\theta_3 - \theta_4 - \text{asin}(n_x c_{\theta_1} s_{\theta_2} - n_z c_{\theta_2} + n_y s_{\theta_1} s_{\theta_2}) \\ \pi - \theta_4 - \theta_3 - \text{asin}(n_x c_{\theta_1} s_{\theta_2} - n_z c_{\theta_2} + n_y s_{\theta_1} s_{\theta_2}). \end{cases}$$

We spent quite some time trying to implement these equations to control the model in Gazebo, but unfortunately we did not get it to work. This lead to looking into ROS packages that could solve it for us. After failing to integrate both MoveIt! and IKFast, we decided to go for another solution. Robotics System Toolbox, a add-on for MATLAB, can solve the inverse kinematics when given a URDF file, so we decided to create a MATLAB node which solved the inverse kinematics. The downside with this approach is that is a numerical solver, meaning it is quite slow.

4.6 D-H parameters

To get a model with the same link- and joint parameters as the physical robot, we measured the robot and made a new URDF model. The final model can be seen in figure 15, whilst the D-H parameters can be seen in table 5.

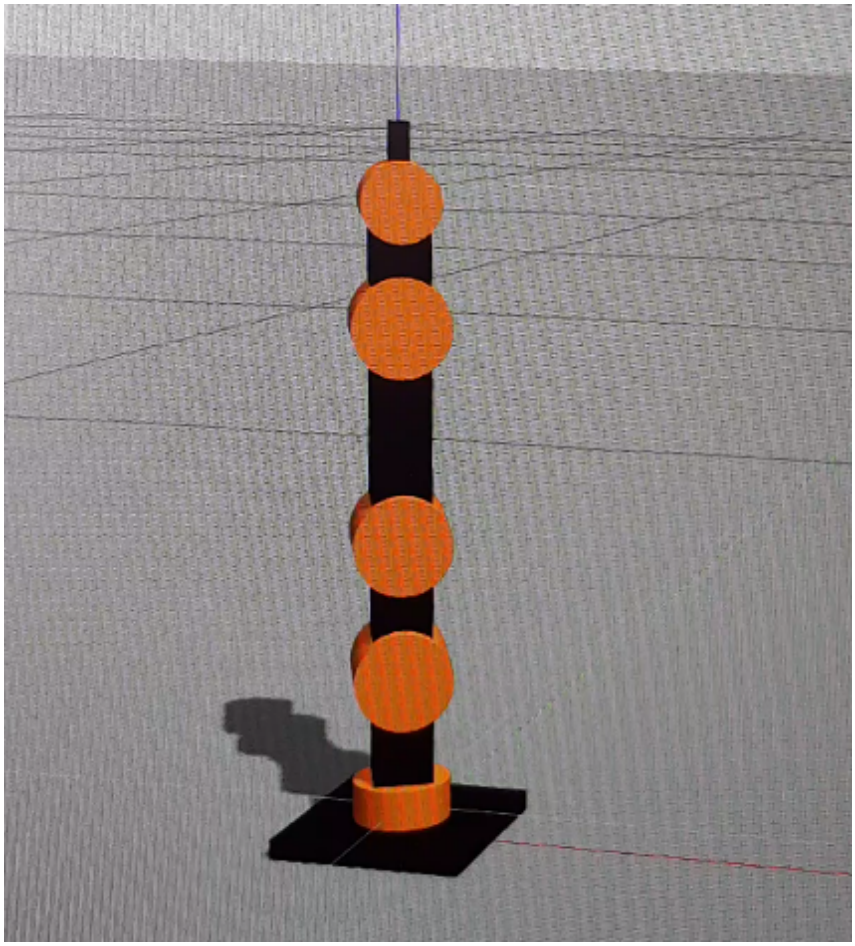


Figure 15: Final model

D-H parameters				
Link	a	α	d	θ
1	0	0	0	θ_1
2	0	90	3 cm	θ_2
3	7.5 cm	0	0	θ_3
4	11.5 cm	0	0	θ_4
5	7.3 cm	0	0	θ_5
6	2.5 cm	0	0	0

Table 5: D-H table

4.7 Synchronization

The next step to the process was to synchronize the speed of the real robot and the simulated model. In this experiment, we ran the physical robot alongside with the Gazebo model and adjusted parameters to get synchronized movement. The result of this case study was a desired 1:1 relationship between the physical robotic arm and the model in Gazebo. Figure 16 shows the setup used to compare the speed of the model and the physical robot.

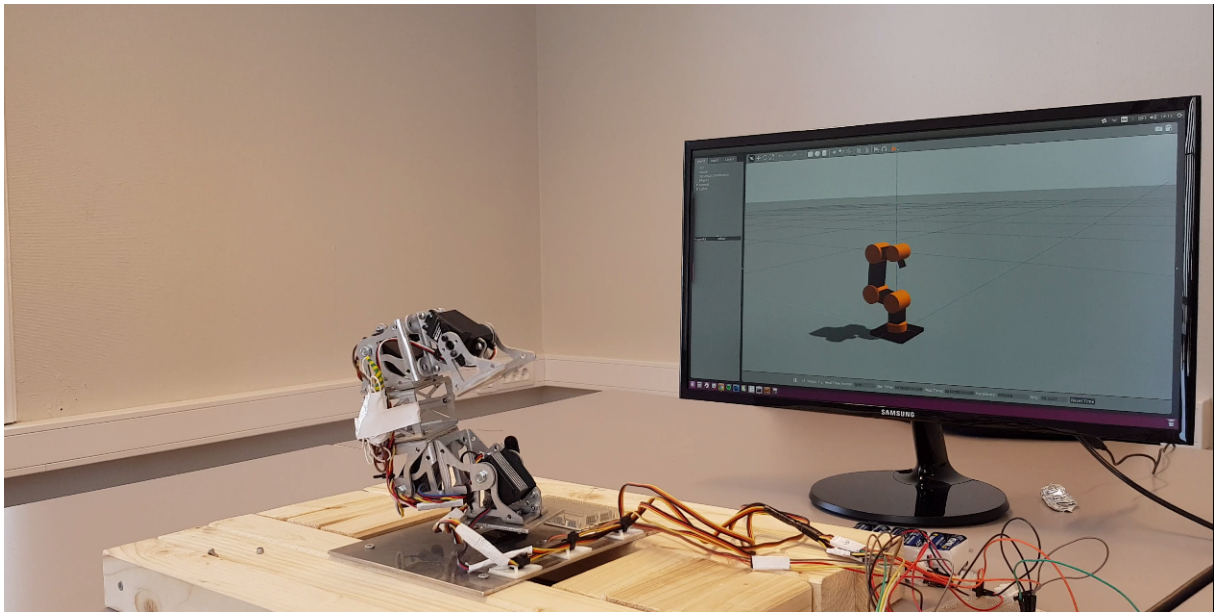


Figure 16: Synchronization

4.8 Final case study

The final case study of ARPA included setting up the system in its entirety. The main goal was to place red markers around in the workspace of ARPA, then send a ready signal to the controller and allow ARPA move to the red marker.

During this final case study, the middle servo motor started operating sub-optimal. It seemed that it got weaker and could not handle the torque the way it had been doing in earlier experiments. We were still able to make ARPA touch the red marker, and were satisfied with the result of the final case study. Figure 17 shows the setup with the different views of the final case study.

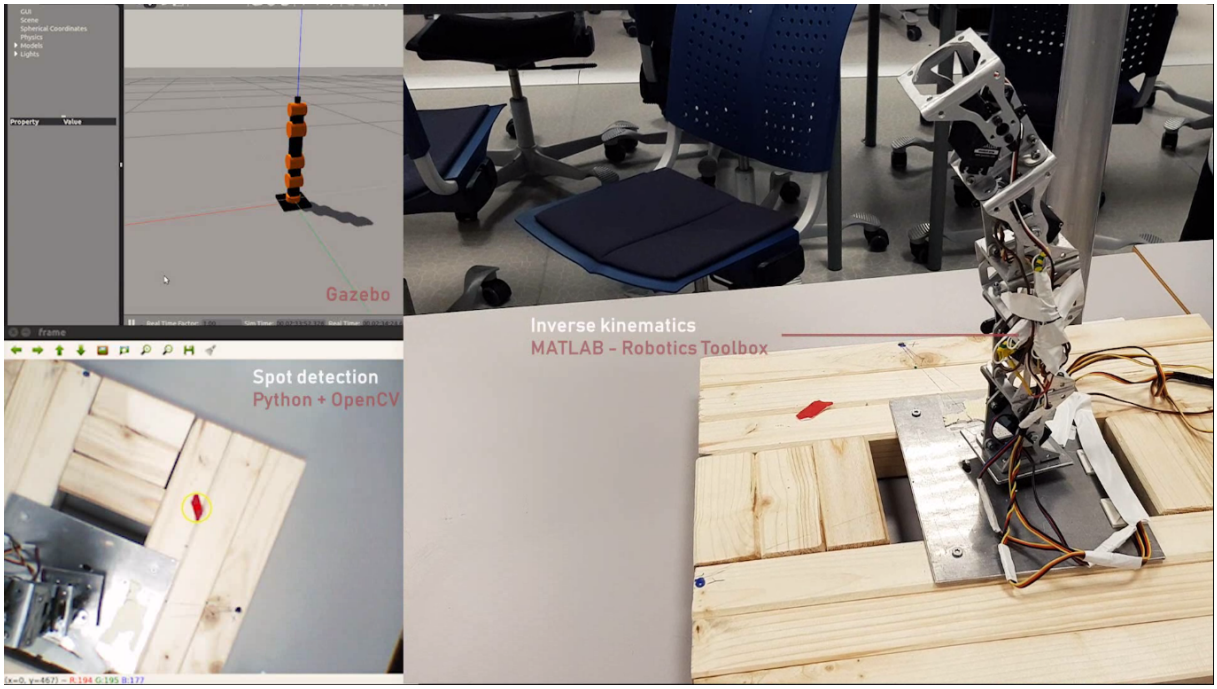


Figure 17: Final case study

5 Discussion

The short summary of our final thoughts about the project is that we are happy with the end result. In hindsight the biggest challenge except for the kinematics which will be discussed later, was numerous hardware faults. Naturally, as we are two software students with limited knowledge in terms of the mechanical and electrical facets of the project, some of the challenges we met might be self-inflicted.

When handed the project we were of the impression that the physical system was well tested and would "just work". As a result of this rather naive mindset, the mechanical and electrical development of the project ended up having much bigger scopes than initially planned in the project. The following paragraphs will discuss the challenges met during the project and is meant to serve as a heads-up for future student development.

5.1 Challenges

Malfunctioning servos

The initial servos on the manipulator were powered through a single Arduino Mega2560 so it is hard to conclude that these were at fault. In total, there was one servo-fatality during the project and we suspect that the servo in the third joint is soon to follow. When changing the power supply to the 12 AA batteries together with two Arduino Megas, we noticed a big difference in actuation from the servos but they still seem to tire. One of the theories we have discussed is that the construction of the robot is sub-optimal, thus inflicting unnecessary strain, especially to the bottom three servos.

Sub-optimal construction/assembly

At first sight, the robot is very skewed, undoubtedly causing strain to the servos, as aforementioned. This especially strains the bottom servo, which carries the total weight on the single rotational pin on the servo, with no support. This might be a reason for it's misbehaving and constant wiggling.

There are also problems with friction in the assembly of the servos on the manipulator and also between the links. If the screws are tightened to the point where the robot feels stable, the

friction generated between the far side of the servos and the actual mounting bracket on the link leaves the servo immobile. The other option is tightening the screws so that the servos are able to move, this leaves the robot in a more dodgy physical state, with a lot of unwanted rotation and movement, screws also tend to fall out while running the robot.

In terms of unwanted rotation and movement the servos seems to misplace after running a while. When initially calibrated to be perpendicular to the rig the servos seem to misplace more and more, which demands recalibration more often than what should be expected.

We also experienced a challenge of the robot falling over and solved this by making the rig on which it now is mounted to. Another benefit of the rig, is that it raises the manipulator's actionable area to approximately the same height as the already mounted metal plate that the robot itself is mounted to. The current camera rig is also sub-optimal, to put it nicely.

Sub-optimal power supply

Although the servos get enough power through the 12 AA batteries and two Arduinos, the solution should not be a permanent one as the batteries are burned quickly. In total the robot has consumed a total of 48 AA batteries. A more permanent solution should be to facilitate for a power supply, during this project we did not want to dabble in this due to a lack of knowledge.

"Cheating" with the kinematics

The kinematics proved a bigger challenge than first anticipated, and the work done ended up being a proof of work, unused in the actual project. Although this was a major setback, both in terms of time used and on a personal scale, we still managed to get a working kinematic solver with the help of Matlab. We also tried ROS MoveIt, but could not get it to give meaningful results.

5.2 Summary

As stated in the introduction to this section, we are happy with the end result of the project. Seen in regards to the goals we set we reached all goals except for having our own functioning kinematic model. The positive side to this is that we still managed to get a working product.

The first goal which was to gain knowledge in ROS, is in our opinion reached in terms of the project's size. We feel that the delivered ROS-system is set up in a logical and tidy manner, where we have been able to learn and implement many of ROS' capabilities and functions.

In terms of the stated goal of developing a fully functional prototype, we are happy with the final delivery and demo, apart from the kinematics which is slow. The integration of the system was a process smoother than expected, especially the integration of the final part which was the detection system. We managed to get a approximate 1:1 relationship between the physical and virtual model, apart from physical defects and other issues arising during the project. We also managed to integrate the detection system so the robot acts based on the object detection. We also managed to develop the kinematic model for ARPA and although it ended up being unused, the work done gave good insights in the mathematics behind robotic control.

The project as a whole has been a positive experience although the learning curve seemed steep at times and many unexpected challenges arose during the project. Both participants of the project are very pleased with both the end result and the learning outcome and the project has definitely sparked a greater interest in the field of robotics.

Appendices

A Complete kinematic solution

Forward kinematics:

DH-parameters are given in table **??**:

Link	a	α	d	θ
1	0	0	d_1	θ_1
2	0	90°	d_2	θ_2
3	a_3	0	0	θ_3
4	a_4	0	0	θ_4
5	a_5	0	0	θ_5
6	a_6	0	0	0

Transformation matrix T_i^{i-1} is specified as:

$$T_i^{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & a_i \\ s_{\theta_i}c_{\alpha_i} & c_{\theta_i}c_{\alpha_i} & s_{\alpha_i} & -s_{\alpha_i}d_i \\ s_{\theta_i}s_{\alpha_i} & c_{\theta_i}s_{\alpha_i} & c_{\alpha_i} & c_{\alpha_i}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

With the input of the DH-parameters, the following matrices are derived:

$$A_1 = \begin{bmatrix} c_{\theta_1} & -s_{\theta_1} & 0 & 0 \\ s_{\theta_1} & c_{\theta_1} & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, A_2 = \begin{bmatrix} c_{\theta_2} & -s_{\theta_2} & 0 & 0 \\ 0 & 0 & -1 & -d_2 \\ s_{\theta_2} & c_{\theta_2} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} c_{\theta_3} & -s_{\theta_3} & 0 & a_3 \\ s_{\theta_3} & c_{\theta_3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, A_4 = \begin{bmatrix} c_{\theta_4} & -s_{\theta_4} & 0 & a_4 \\ s_{\theta_4} & c_{\theta_4} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} c_{\theta_5} & -s_{\theta_5} & 0 & a_5 \\ s_{\theta_5} & c_{\theta_5} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, A_6 = \begin{bmatrix} 1 & 0 & 0 & a_6 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_0^6 gives a set of linear equations for forward kinematics:

$$T_H^R = T_0^6 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_1 A_2 A_3 A_4 A_5 A_6$$

$$= \begin{bmatrix} c_{\theta_{2345}} c_{\theta_1} & -s_{\theta_{2345}} c_{\theta_1} & s_{\theta_1} & d_2 s_{\theta_1} + a_6 c_{\theta_{2345}} c_{\theta 1} + a_4 c_{\theta_{23}} c_{\theta_1} + a_3 c_{\theta_1} c_{\theta_2} + a_5 c_{\theta_{234}} c_{\theta_1} \\ c_{\theta_{2345}} s_{\theta 1} & -s_{\theta_{2345}} s_{\theta_1} & -c_{\theta_1} & a_6 c_{\theta_{2345}} s_{\theta_1} - d_2 c_{\theta 1} + a_4 c_{\theta_{23}} s_{\theta_1} + a_3 c_{\theta_2} s_{\theta 1} + a_5 c_{\theta_{234}} s_{\theta_1} \\ s_{\theta_{2345}} & c_{\theta_{2345}} & 0 & d_1 + a_6 s_{\theta_{2345}} + a_4 s_{\theta_{23}} + a_3 s_{\theta_2} + a_5 s_{\theta_{234}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse kinematics:

The full set of equations for the inverse kinematics are summarized at the end of this section.

$$(T_1^0)^{-1}T_6^0 = T_2^1T_3^2T_4^3T_5^4T_6^5$$

$$\Rightarrow A_1^{-1} \times \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_2A_3A_4A_5A_6$$

$$\Rightarrow \begin{bmatrix} c_{\theta_1} & s_{\theta_1} & 0 & 0 \\ -s_{\theta_1} & c_{\theta_1} & 0 & 0 \\ 0 & 0 & 1 & -d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_2A_3A_4A_5A_6$$

$$\Rightarrow \begin{bmatrix} n_xc_{\theta_1} + n_ys_{\theta_1} & o_xc_{\theta_1} + o_ys_{\theta_1} & a_xc_{\theta_1} + a_ys_{\theta_1} & p_xc_{\theta_1} + p_ys_{\theta_1} \\ n_yc_{\theta_1} - n_xs_{\theta_1} & o_yc_{\theta_1} - o_xs_{\theta_1} & a_yc_{\theta_1} - a_xs_{\theta_1} & p_yc_{\theta_1} - p_xs_{\theta_1} \\ n_z & o_z & a_z & p_z - d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_{\theta_{2345}} & -s_{\theta_{2345}} & 0 & a_6c_{\theta_{2345}} + a_4c_{\theta_{23}} + a_3c_{\theta_2} + a_5c_{\theta_{234}} \\ 0 & 0 & -1 & -d_2 \\ s_{\theta_{2345}} & c_{\theta_{2345}} & 0 & a_6s_{\theta_{2345}} + a_4s_{\theta_{23}} + a_3s_{\theta_2} + a_5s_{\theta_{234}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equating elements at *(2, 4)* to find θ_1, and elements at *(3, 4)* to find θ_4:

$$p_yc_{\theta_1} - p_xs_{\theta_1} = -d_2$$

$$\Rightarrow \theta_1 = 2\text{atan}\left(\frac{p_x \pm \sqrt{-d_2^2 + p_x^2 + p_y^2}}{d_2 - p_y}\right). \tag{1}$$

$$p_z - d_1 = a_6 s_{\theta_{2345}} + a_4 s_{\theta_{23}} + a_3 s_{\theta_2} + a_5 s_{\theta_{234}}$$

$$\Rightarrow p_z - d_1 = a_6 n_z + a_4 s_{\theta_{23}} + a_3 s_{\theta_2} + a_5 s_{\theta_{234}} \tag{2}$$

$$\Rightarrow \theta_4 = \begin{cases} -\theta_2 - \theta_3 - \operatorname{asin}\left(\frac{d_1 - p_z + a_6 n_z + a_4 s_{\theta_{23}} + a_3 s_{\theta_2}}{a_5}\right) \\ \pi - \theta_2 - \theta_3 - \operatorname{asin}\left(\frac{d_1 - p_z + a_6 n_z + a_4 s_{\theta_{23}} + a_3 s_{\theta_2}}{a_5}\right). \end{cases}$$

$$(T_2^0)^{-1}T_6^0 = T_3^2T_4^3T_5^4T_6^5$$

$$\Rightarrow (A_1A_2)^{-1} \times \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_3A_4A_5A_6$$

$$\Rightarrow \begin{bmatrix} c_{\theta_1}c_{\theta_2} & c_{\theta_2}s_{\theta_1} & s_{\theta_2} & -d_1s_{\theta_2} \\ -c_{\theta_1}s_{\theta_2} & -s_{\theta_1}s_{\theta_2} & c_{\theta_2} & -d_1c_{\theta_2} \\ s_{\theta_1} & -c_{\theta_1} & 0 & -d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_3A_4A_5A_6$$

$$\Rightarrow \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \end{bmatrix} = \begin{bmatrix} c_{\theta_{345}} & -s_{\theta_{345}} & 0 & a_3 + a_5c_{\theta_{34}} + a_4c_{\theta_3} + a_6c_{\theta_{345}} \\ s_{\theta_{345}} & c_{\theta_{345}} & 0 & a_5s_{\theta_{34}} + a_4s_{\theta_3} + a_6s_{\theta_{345}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

$$r_{11} = n_z s_{\theta_2} + n_x c_{\theta_1} c_{\theta_2} + n_y c_{\theta_2} s_{\theta_1}$$

$$r_{12} = o_z s_{\theta_2} + o_x c_{\theta_1} c_{\theta_2} + o_y c_{\theta_2} s_{\theta_1}$$

$$r_{13} = a_z s_{\theta_2} + a_x c_{\theta_1} c_{\theta_2} + a_y c_{\theta_2} s_{\theta_1}$$

$$r_{14} = p_z s_{\theta_2} - d_1 s_{\theta_2} + p_x c_{\theta_1} c_{\theta_2} + p_y c_{\theta_2} s_{\theta_1}$$

$$r_{21} = n_z c_{\theta_2} - n_x c_{\theta_1} s_{\theta_2} - n_y s_{\theta_1} s_{\theta_2}$$

$$r_{22} = o_z c_{\theta_2} - o_x c_{\theta_1} s_{\theta_2} - o_y s_{\theta_1} s_{\theta_2}$$

$$r_{23} = a_z c_{\theta_2} - a_x c_{\theta_1} s_{\theta_2} - a_y s_{\theta_1} s_{\theta_2}$$

$$r_{24} = p_z c_{\theta_2} - d_1 c_{\theta_2} - p_x c_{\theta_1} s_{\theta_2} - p_y s_{\theta_1} s_{\theta_2}$$

$$r_{31} = n_x s_{\theta_1} - n_y c_{\theta_1}$$

$$r_{32} = o_x s_{\theta_1} - o_y c_{\theta_1}$$

$$r_{33} = a_x s_{\theta_1} - a_y c_{\theta_1}$$

$$r_{34} = p_x s_{\theta_1} - p_y c_{\theta_1} - d_2$$

$$r_{41} = 0$$

$$r_{42} = 0$$

$$r_{43} = 0$$

$$r_{44} = 1$$

Equating elements at *(1, 3)* to find θ_2 and *(2, 1)* to find θ_5:

$$a_z s_{\theta_2} + a_x c_{\theta_1} c_{\theta_2} + a_y c_{\theta_2} s_{\theta_1} = 0$$

$$\Rightarrow \theta_2 = 2\,\text{atan}\left(\frac{a_z \pm \sqrt{a_x^2 c_{\theta_1}^2 + 2a_x a_y c_{\theta_1} s_{\theta_1} + a_y^2 s_{\theta_1}^2 + a_z^2}}{a_x c_{\theta_1} + a_y s_{\theta_1}}\right).$$

$$n_z c_{\theta_2} - n_x c_{\theta_1} s_{\theta_2} - n_y s_{\theta_1} s_{\theta_2} = s_{\theta_{345}}$$

$$\Rightarrow \theta_5 = \begin{cases} -\theta_3 - \theta_4 - \text{asin}(n_x c_{\theta_1} s_{\theta_2} - n_z c_{\theta_2} + n_y s_{\theta_1} s_{\theta_2}) \\ \\ \pi - \theta_4 - \theta_3 - \text{asin}(n_x c_{\theta_1} s_{\theta_2} - n_z c_{\theta_2} + n_y s_{\theta_1} s_{\theta_2}). \end{cases}$$

$$(T_3^0)^{-1}T_6^0 = T_4^3T_5^4T_6^5$$

$$\Rightarrow (A_1A_2A_3)^{-1} \times \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_4A_5A_6$$

$$\Rightarrow \begin{bmatrix} c_{\theta_{23}}c_{\theta_1} & c_{\theta_{23}}s_{\theta_1} & s_{\theta_{23}} & -d_1s_{\theta_{23}} - a_3c_{\theta_3} \\ -s_{\theta_{23}}c_{\theta_1} & -s_{\theta_{23}}s_{\theta_1} & c_{\theta_{23}} & a_3s_{\theta_3} - d_1c_{\theta_{23}} \\ s_{\theta_1} & -c_{\theta_1} & 0 & -d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_4A_5A_6$$

$$\Rightarrow \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \end{bmatrix} = \begin{bmatrix} c_{\theta_{45}} & -s_{\theta_{45}} & 0 & a_4 + a_6c_{\theta_{45}} + a_5c_{\theta_4} \\ s_{\theta_{45}} & c_{\theta_{45}} & 0 & a_6s_{\theta_{45}} + a_5s_{\theta_4} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

$$\begin{aligned} r_{13} &= a_zc_{\theta_2}s_{\theta_3} + a_zc_{\theta_3}s_{\theta_2} + a_xc_{\theta_1}c_{\theta_2}c_{\theta_3} + a_yc_{\theta_2}c_{\theta_3}s_{\theta_1} - a_xc_{\theta_1}s_{\theta_2}s_{\theta_3} - a_ys_{\theta_1}s_{\theta_2}s_{\theta_3} \\ &= a_z(c_{\theta_2}s_{\theta_3} + c_{\theta_3}s_{\theta_2}) + c_{\theta_2}c_{\theta_3}(a_xc_{\theta_1} + a_ys_{\theta_1}) - s_{\theta_2}s_{\theta_3}(a_xc_{\theta_1} + a_ys_{\theta_1}) \end{aligned}$$

Equating elements at *(1, 3)* to find θ_3:

$$\Rightarrow \theta_3 = \begin{cases} -2\,\text{atan}\left(\frac{\sqrt{a_x^2c_{\theta_1}^2 + s_{2\theta_1}a_xa_y - a_y^2c_{\theta_1}^2 + a_y^2 + a_z^2} - a_zc_{\theta_2} + a_xc_{\theta_1}s_{\theta_2} + a_ys_{\theta_1}s_{\theta_2}}{a_zs_{\theta_2} + a_xc_{\theta_1}c_{\theta_2} + a_yc_{\theta_2}s_{\theta_1}}\right) \\ 2\,\text{atan}\left(\frac{\sqrt{a_x^2c_{\theta_1}^2 + s_{2\theta_1}a_xa_y - a_y^2c_{\theta_1}^2 + a_y^2 + a_z^2} + a_zc_{\theta_2} - a_xc_{\theta_1}s_{\theta_2} - a_ys_{\theta_1}s_{\theta_2}}{a_zs_{\theta_2} + a_xc_{\theta_1}c_{\theta_2} + a_yc_{\theta_2}s_{\theta_1}}\right) \end{cases}$$

Angles θ_1, θ_2, θ_3, θ_4, θ_5 are given by the following equations:

$$\theta_1 = 2\text{atan}\left(\frac{p_x \pm \sqrt{-d_2^2 + p_x^2 + p_y^2}}{d_2 - p_y}\right).$$

$$\theta_2 = 2\,\text{atan}\left(\frac{a_z \pm \sqrt{a_x^2 c_{\theta_1}^2 + 2a_x a_y c_{\theta_1} s_{\theta_1} + a_y^2 s_{\theta_1}^2 + a_z^2}}{a_x c_{\theta_1} + a_y s_{\theta_1}}\right)$$

$$\theta_3 = \begin{cases} -2\,\text{atan}\left(\frac{\sqrt{a_x^2 c_{\theta_1}^2 + s_{2\theta_1} a_x a_y - a_y^2 c_{\theta_1}^2 + a_y^2 + a_z^2} - a_z c_{\theta_2} + a_x c_{\theta_1} s_{\theta_2} + a_y s_{\theta_1} s_{\theta_2}}{a_z s_{\theta_2} + a_x c_{\theta_1} c_{\theta_2} + a_y c_{\theta_2} s_{\theta_1}}\right) \\ 2\,\text{atan}\left(\frac{\sqrt{a_x^2 c_{\theta_1}^2 + s_{2\theta_1} a_x a_y - a_y^2 c_{\theta_1}^2 + a_y^2 + a_z^2} + a_z c_{\theta_2} - a_x c_{\theta_1} s_{\theta_2} - a_y s_{\theta_1} s_{\theta_2}}{a_z s_{\theta_2} + a_x c_{\theta_1} c_{\theta_2} + a_y c_{\theta_2} s_{\theta_1}}\right) \end{cases}$$

$$\theta_4 = \begin{cases} -\theta_2 - \theta_3 - \text{asin}\left(\frac{d_1 - p_z + a_6 n_z + a_4 s_{\theta_{23}} + a_3 s_{\theta_2}}{a_5}\right) \\ \pi - \theta_2 - \theta_3 - \text{asin}\left(\frac{d_1 - p_z + a_6 n_z + a_4 s_{\theta_{23}} + a_3 s_{\theta_2}}{a_5}\right). \end{cases}$$

$$\theta_5 = \begin{cases} -\theta_3 - \theta_4 - \text{asin}(n_x c_{\theta_1} s_{\theta_2} - n_z c_{\theta_2} + n_y s_{\theta_1} s_{\theta_2}) \\ \pi - \theta_4 - \theta_3 - \text{asin}(n_x c_{\theta_1} s_{\theta_2} - n_z c_{\theta_2} + n_y s_{\theta_1} s_{\theta_2}). \end{cases}$$

B GitHub repository

Link to GitHub: https://github.com/andrehaland/DFSM3200-robotic-arm

References

[1] Aslak Berge. Avhengig av polske kniver. *http://ilaks.no/avhengig-av-polske-kniver-2/*. Aug 2018.

[2] Hanne Digre. Lønnsom foredling av sjømat i Norge. 2014.

[3] Abdellatif Baba.Robot arm control with Arduino. June 2017.

[4] Jeffry Walker, Joshua Beverly, Christopher Cook, Migueal de Rojas. Anthropomorphic Robotic Manipulator (ARM). 2012

[5] Open Source Robotics Foundation. Understanding ROS Nodes. *http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes*. Aug 2017.

[6] Arduino. Arduino libraries. *https://www.arduino.cc/en/Reference/Libraries*

[7] Lentin Joseph. Mastering ROS for Robotics Programming. Des 2015