ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Jan VÁVERKA** |
| Osobní číslo: | **A20N0038P** |
| Studijní program: | **N3918 Aplikované vědy a informatika** |
| Studijní obor: | **Kybernetika a řídicí technika** |
| Téma práce: | **Design of a Cable-Driven parallel manipulator for pick&place applications** |
| Zadávající katedra: | **Katedra kybernetiky** |

## Zásady pro vypracování

1. Study the possibilities of creating kinematic models of Cable-Driven parallel robots.
2. Study the solution of the robot's kinematic tasks.
3. Design the architecture of a Cable-Driven robot for positioning the load in the XYZ space.
4. Implement the manipulator model in the Matlab/Simulink/SimMechanics environment.
5. Implement a trajectory generator – movement between points of interest.
6. Discuss the results, benefits and limitations of the designed robot and it's control.

Rozsah diplomové práce: **40 – 50 stránek A4**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:


1. Bruckmann, Tobias & Mikelsons, Lars & Brandt, Thorsten & Hiller, Manfred & Schramm, Dieter. (2008). Wire Robots Part I: Kinematics, Analysis & Design.
2. Bruckmann, Tobias & Mikelsons, Lars & Brandt, Thorsten & Hiller, Manfred & Schramm, Dieter. (2008). Wire Robots Part II: Dynamics, Control & Application.
3. Morris, Melissa & Shoham, Moshe. (2009). Applications and Theoretical Issues of Cable-Driven Robots.
4. Gosselin, Clément. (2013). Cable-driven parallel mechanisms: state of the art and perspectives.
5. Švejda, Martin. (2016). Optimalizace Robotických Architektur.
6. Sciavicco, Lorenzo. (2005). Modelling and Control of Robot Manipulators.
7. Abdolshah, Saeed. (2016). Trajectory Planning and Control of Cable-Driven Parallel Robots.


Vedoucí diplomové práce: **Ing. Martin Švejda, Ph.D.**
Výzkumný program 1

Datum zadání diplomové práce: **1. října 2021**
Termín odevzdání diplomové práce: **23. května 2022**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

L.S.

**Doc. Ing. Miloš Železný, Ph.D.**
děkan

**Prof. Ing. Josef Psutka, CSc.**
vedoucí katedry

V Plzni dne 1. října 2021

# PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě diplomovou/bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou/diplomovou práci vypracoval(a) samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 18.5.2022 .....

**Acknowledgement**

I would like to thank Ing. Martin Švejda PhD. for allowing me to work on this topic and providing me the resources to make the project more affordable. Additionally I would like to thank Bond for fetching some of the components needed to construct the manipulator.

**Anotace**

V této práci jsou popsaný paralelní kabelový robot z teoretického i praktického hlediska, včetně analýzy pracovního prostoru, dopředné a zpětné kinematické úlohy, generování trajektorie, softwarového modelování a konstrukce hardwarového fyzikálního modelu.

**Klíčová slova**

paralelní robot, kabelový robot, kinematické úlohy, zpětná kinematická úloha, dopředná kinematická úloha, plánování trajektorie, pick&place manipulátor, zájmové body

**Abstract**

In this thesis the theory and construction of a wire robot is discussed, including the analysis of the workspace, inverse and forward kinematic tasks, trajectory generation, software model and constructing a physical model to show the theory in the real world.

**Key words**

parallel wire robots, cable robots, kinematic tasks, inverse kinematics, forward kinematics, trajectory planning, pick and place manipulator, points of interest

# Contents

# Chapter 1

# Introduction

Nowadays all kinds of robots are broadly used in our society for different purposes. Common in the industrial environment are manipulators. They are used to move the end effector (usually some kind of a tool) between points of interest over a planned trajectory. The end effector tool can interact with the environment and thus the manipulator is able to complete simple tasks autonomously or in a cooperation with an operator. Robotic manipulators are more efficient, accurate and after the initial investment cheaper to maintain at their respective tasks than human workers. There is a wide range of tools that can be used as an end effector, starting with a simple screwdriver or a brush and ending with a gripper or a suction cup that can be used to transport any load that's needed. Because of that, the possible tasks that robotic manipulators can complete are basically infinite.

Parallel robot is a manipulator that connects the end effector to the base by several actuators. They have a smaller workspace compared to serial manipulators, however they are usually stiffer and can achieve given point of interest with a higher accuracy and repeatability.

Finally a parallel wire robot is a parallel robot that is connected to the base by wires/cables. The position (and orientation) of the end effector is achieved by pulling or releasing the wires usually connected to actuated spools. This solution is cost-effective, safe and allows for a bigger workspace compared to classical parallel robots thanks to the possibility of coiling the wires.

# Chapter 2

# Theory

## 2.1 Analysis of the workspace

The main restriction of parallel robots is their limited workspace. Classical parallel robots with rigid legs often have a small workspace limited by how far the rigid legs can extend and contract. One example of this behaviour would be the most well-known representative of parallel robots, the Steward platform. Though Steward platform can support heavy loads, it's typically used in applications where the desired motion mostly consists of changes in the rotation of the platform rather than its position.

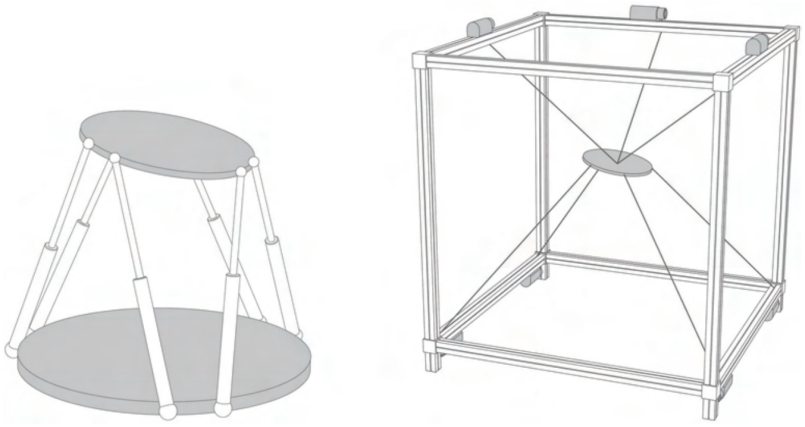


Figure 2.1: The difference between a classical parallel robot (Steward platform on the left side) and a parallel wire robot (right side) - illustration taken from [1]

Since wire robots don't have rigid legs, their workspace should naturally be larger. However by replacing the legs by non-rigid wires, only pulling forces are available. That creates a new unilateral constraints on the wires and makes the construction of the robot more complicated. This creates an outer shell of the wire robot workspace, since no pulling force can move the efector out of the area defined by the wire spool ends. Since the wires are not rigid links by themselves, a high enough force has to be applied at all times to compensate the flexibility of the wires. The acting force on each of the wires is actually the main constraint to calculate the possible workspace of the parallel wire robot (also known as the force equilibrium) and it further limits the workspace. This external force can be increased by wires pulling on the opposite direction of the observed wire, however the shape and weight of the efector is also a big factor when calculating the workspace.

One of the possible way to calculate and plot the workspace is to consider all the possible positions of the end efector and calculate the force acting on each of the wires in that specific steady state ($F_r$). If all of the wires satisfy the minimal ($F_{min}$) and maximal ($F_{max}$) force that is allowed on the wire, we can consider that point as a part of the force equilibrium workspace.

$$F_{min} < F < F_{max} \tag{2.1}$$

After inspecting all of the grid points described by the robot geometry, we can plot the points which are a part of the workspace. As an example my setup consists of 4 wires which have their spools positioned in the upper corners of a 0.5 meter cube. The considered efector is a small cylinder with the mass of $m_e = 0.25kg$. In figure (2.2) the visualisation of the workspace can be observed.

It is apparent that the final force equilibrium is quite small compared to the dimensions of the cube towards the bottom. That is caused by not having any wires on the bottom side of the robot, which means the external forces consist of mostly just the gravitational force. One way to increase the workspace would be to add another set of actuated spools with wires to the bottom of the robot. This solution would not only increase the workspace but also increase the maximal speeds which we could achieve. However this solution would more or less double the cost of the robot and could not always be possible.

Another possibility would be to increase the mass of the end efector by for example using additional weight. This is of course dependant on the available torque of the actuated spools. Additionally it would of course lower the possible speed limits of the robot. However it can be a cost-effective

solution for smaller applications. If we increase the weight of the efector to $m_e = 1kg$, we can already see that the size of the workspace has increased dramatically.

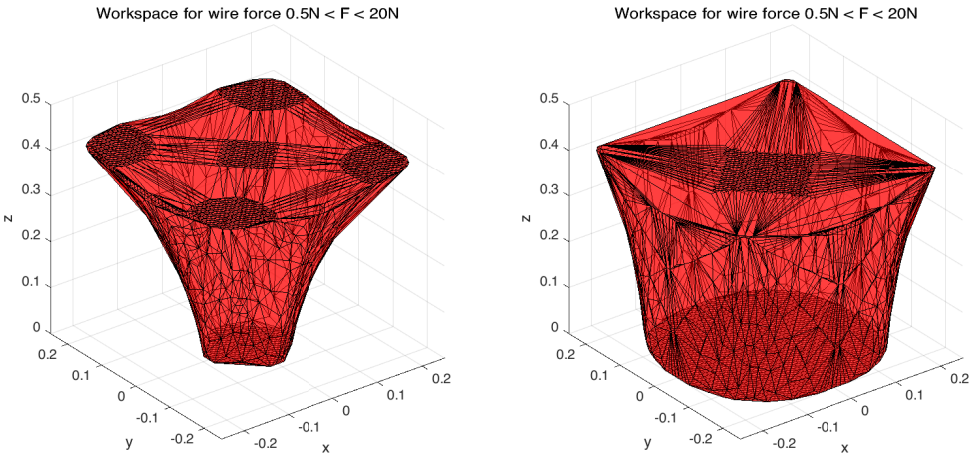



Figure 2.2: Visualisation of the force equilibrium workspace of a 4-wire robot with a cylinder efector with a mass of $m_e = 0.25kg$ (left) or $m_e = 1kg$ (right)

Apart from the discrete analysis of the workspace, continuous interval methods are available as well to compute the force equilibrium workspace of the parallel wire robot. Such methods are based on solving the constraint satisfaction problem and finding the solution (workspace). These methods are further described in [1].

## 2.2 Inverse kinematic task

The purpose of an inverse kinematic task is to determine the joint variables (in our case the length of the individual wires) of a specific end-efector position in the workspace. Solving this fundamental task is mandatory to precisely control any manipulator. In the case that the kinematic chains connecting the base and the end efector of a parallel manipulator are simple (which is the case for cable-driven manipulators), solving the inverse kinematic task is not as complex as the inverse kinematic task of a serial manipulator. For wires coiled on a spool, the exit point on the spool is dependant on the position of the end efector. If the wires are guided by small eyes or pulleys, the variable exit point can be neglected and only the position of the eye guidance can be used for any position of the end efector. In that case

the inverse kinematic task can be simplified and only a few known perimeters are needed to specify the exact lengths of each leg of the manipulator. To compute the length of a specific leg/wire $\boldsymbol{l}$, the following parameters are needed:

- Constant distance of the guide eye/pulley from the base / world frame $\boldsymbol{b}$
- Constant distance of the wire anchor point from the end efector point $\boldsymbol{p}$
- The desired position of the end efector point in the base coordinate system $\boldsymbol{r}$
- The desired rotation of the end efector point in the base coordinate system $\boldsymbol{R}$

Assuming the wires are stiff enough and tensioned at all times, the length of each wire can by computed by thy Pythagoras theorem:

$$l_i = \|\mathbf{b_i} - \mathbf{r} - \mathbf{R} \cdot \mathbf{p_i}\|_2 \tag{2.2}$$

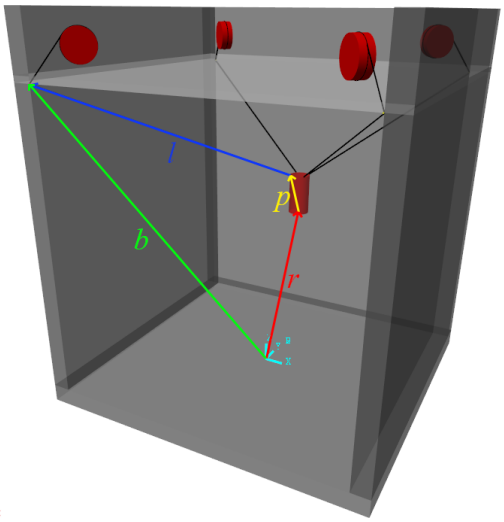



Figure 2.3: Visualisation of the perimeters needed for the inverse kinematic task

## 2.3 Forward kinematic task

Forward kinematic task is a fundamental problem of both serial and parallel manipulators as well. The solution describes the XYZ position **r** and rotation **R** of the end effector in relation to the leg lengths **l**. Unlike the inverse kinematic task, forward kinematics of parallel robots generally do not have an analytical solution and so the solution is often dependant on numerical methods. With eye-point guidance (as described in the inverse kinematics) and tensioned wires, finding the XYZ position of the end efector is equivalent to finding the intersection of **n** spheres (where n is the number of wires) with their center positioned in the eye guidances and their radius equivalent to the lengths of the specific wires.

The solution can be then computed using the Newton-Rhapson solver if the number of DoF is the same as the number of wires. If the number of wires is higher, then an overdetermined system of equations has to be solved (for example a least-square method can be used). Since a numerical approach is required, problems such as finding a correct stable solution arise.

## 2.4 Trajectory generation

Trajectory generation is a task of defining and creating a trajectory/path, which the end-efector of the robot should follow with high accuracy. This trajectory (usually in the XYZ plane) is then transformed using the Inverse Kinematic Task into the robot's coordinate system, which allows for an easy and precise generation of direct commands to the robot's actuators.

The main reason of having a trajectory is to move between the so called "points of interest" and also to avoid any obstacles in the way. For pick&place applications, those points of interest often represent the origin location (where the load is initially placed), the destination (where we want the load to be placed) and optionally any points between those two locations (for example to avoid the obstacles). Usual requirement for the manipulator can be to either stop in the provided point of interest (origin/destination) or to just pass through the point and continue without slowing down.

# Chapter 3

# Simulation Model

## 3.1 Model Structure

First step in creating the final robot was to realize the different options, limitations and objectives but also the overall purpose of the robot. In this case, the main objective was to verify the studied theory and to create an experimental model that would be able to follow the specified planned trajectory to pick and place objects in the robot's workspace. Because of that, the main factor was to reduce the cost but still be able to complete said objectives. For that reason a cubical workspace was used to simplify the construction and make the final parameters of the robot as much symetrical as possible.

Since the ability to control the 3 diferent rotations is not needed in pick and place applications, in theory only 3 wires are needed to be able to move the end efector in a XYZ workspace. That was also how the model was first developed. Since the wires can only act on the end efector with pulling forces, the actuated spools were placed at the top of a 0.5x0.5x0.5 m cube that acted as the robot's workspace. After this decision, the workspace was calculated as described in the previous chapter.

To increase the possible floor area and overall workspace of the robot, it was later decided that an additional actuated spool would be used. This fourth wire doesn't allow us to control the robot in any other degree of freedom, however it does help with increasing the possible workspace and making it more symmetrical as seen in 3.2.

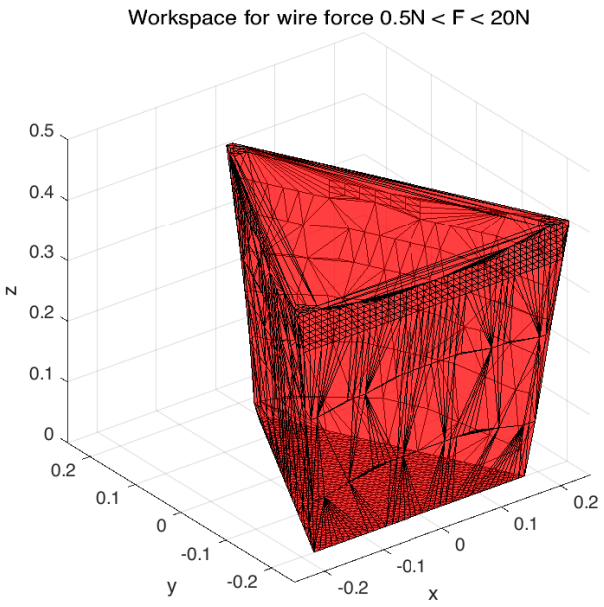



Figure 3.1: The calculated workspace of a cable robot with only 3 wires

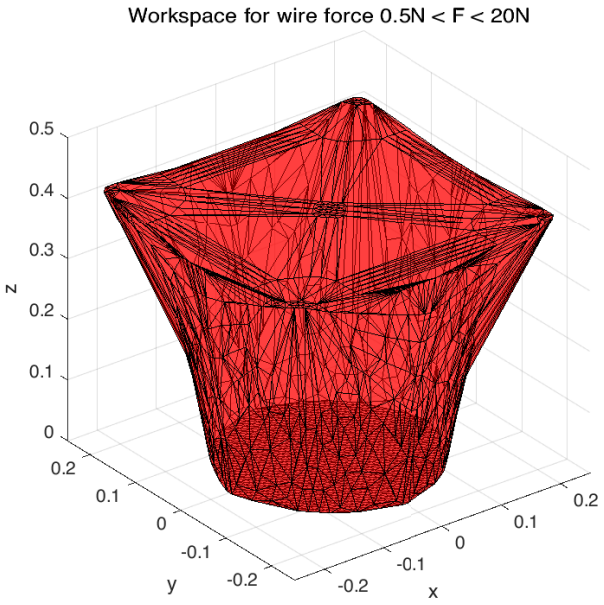



Figure 3.2: The calculated workspace of a cable robot with 4 wires

## 3.2 Modelling Software

For software simulation, Matlab/Simulink environment (R2019A) together with the appropriate Simscape library was used. First the basic world frame + configuration blocks were added together with the visuals of the robot's body. It was decided that a 0.5m x 0.5m x 0.5m XYZ workspace would be used. To accompany for the walls and electronics that would have to be put to the upper part of the robot, the actual height of 0.6 m was used for the walls so that the internal workspace dimensions would match the requirements.

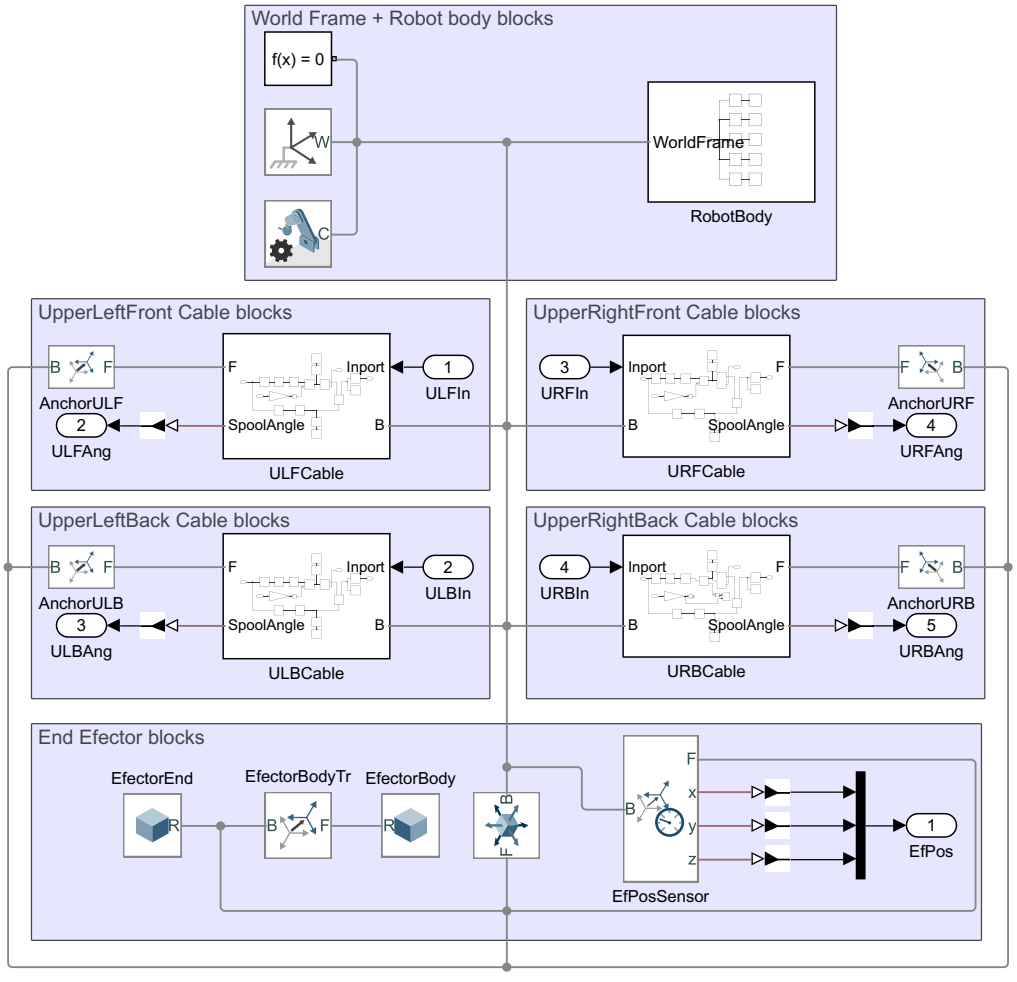



Figure 3.3: The blocks that were used to simulate the robot in the Simulink environment

### 3.2.1 Cables

The actuated spool was simulated by a Belt-Cable Spool block connected to a Revolute Joint block with a desired angle input. The wire was then guided through a small pulley guide that moved the wire exit point to the desired respective location and also helped with the accuracy of the final position of the end efector computed by the inverse kinematic task. The other end of the wire was terminated by the Belt-Cable End block and connected to the respective anchor location on the end efector.

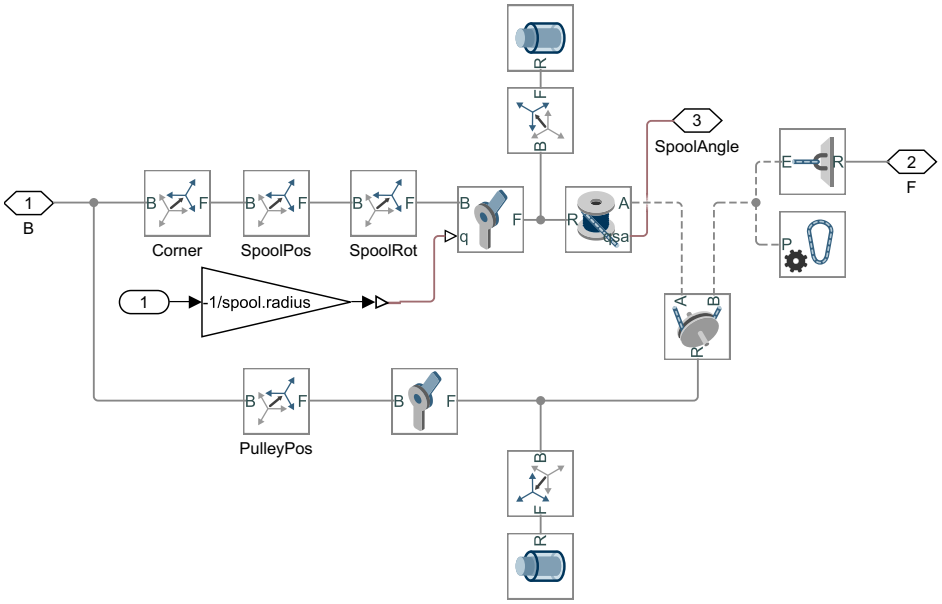



Figure 3.4: The blocks that were used to simulate the actuated spool and wire logic

### 3.2.2 End-efector

A cylindrical end efector with a radius of $r = 11[mm]$ and height $h = 69[mm]$ was used and connected by four wires to the actuated spools. The wires were connected to their respective anchor points on the end efector. The anchors were placed symmetrically at the top part of the cylindrical efector. That way, the top side of the efector is basically a circumscribed circle of a square and the corners of that square are the respective anchors. Since no end efector rotation was required for the application, this anchor placement was sufficient.

To get the coordinates of the anchor placement $p$ relative to the end efector position $r$, the following formulas have to be used:

$$p_{URB} = \left[\frac{r}{\sqrt{2}}, \frac{r}{\sqrt{2}}, h\right] = \left[\frac{11}{\sqrt{2}}, \frac{11}{\sqrt{2}}, 69\right] [mm] \tag{3.1}$$

$$p_{ULB} = \left[-\frac{r}{\sqrt{2}}, \frac{r}{\sqrt{2}}, h\right] = \left[-\frac{11}{\sqrt{2}}, \frac{11}{\sqrt{2}}, 69\right] [mm] \tag{3.2}$$

$$p_{ULF} = \left[-\frac{r}{\sqrt{2}}, -\frac{r}{\sqrt{2}}, h\right] = \left[-\frac{11}{\sqrt{2}}, -\frac{11}{\sqrt{2}}, 69\right] [mm] \tag{3.3}$$

$$p_{URF} = \left[\frac{r}{\sqrt{2}}, -\frac{r}{\sqrt{2}}, h\right] = \left[\frac{11}{\sqrt{2}}, -\frac{11}{\sqrt{2}}, 69\right] [mm] \tag{3.4}$$

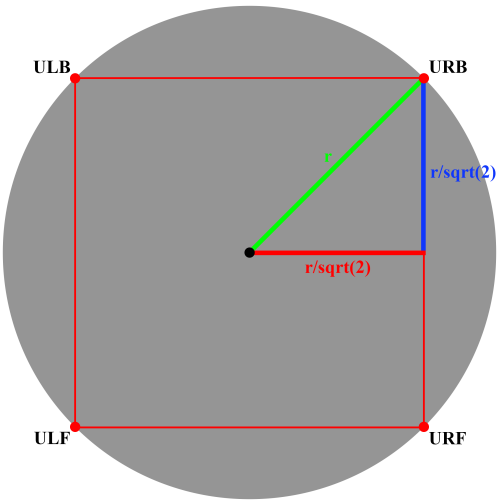



Figure 3.5: The anchor placement and calculation on the top side of the end efector

This type of anchoring help the end efector not to vibrate and wiggle. However one of the disadvantages is that the ende efector rotation is not constant. This can be seen by slight tilting at the edges of the workspace. This could of course be solved by added more cables and henc more degrees of freedom.

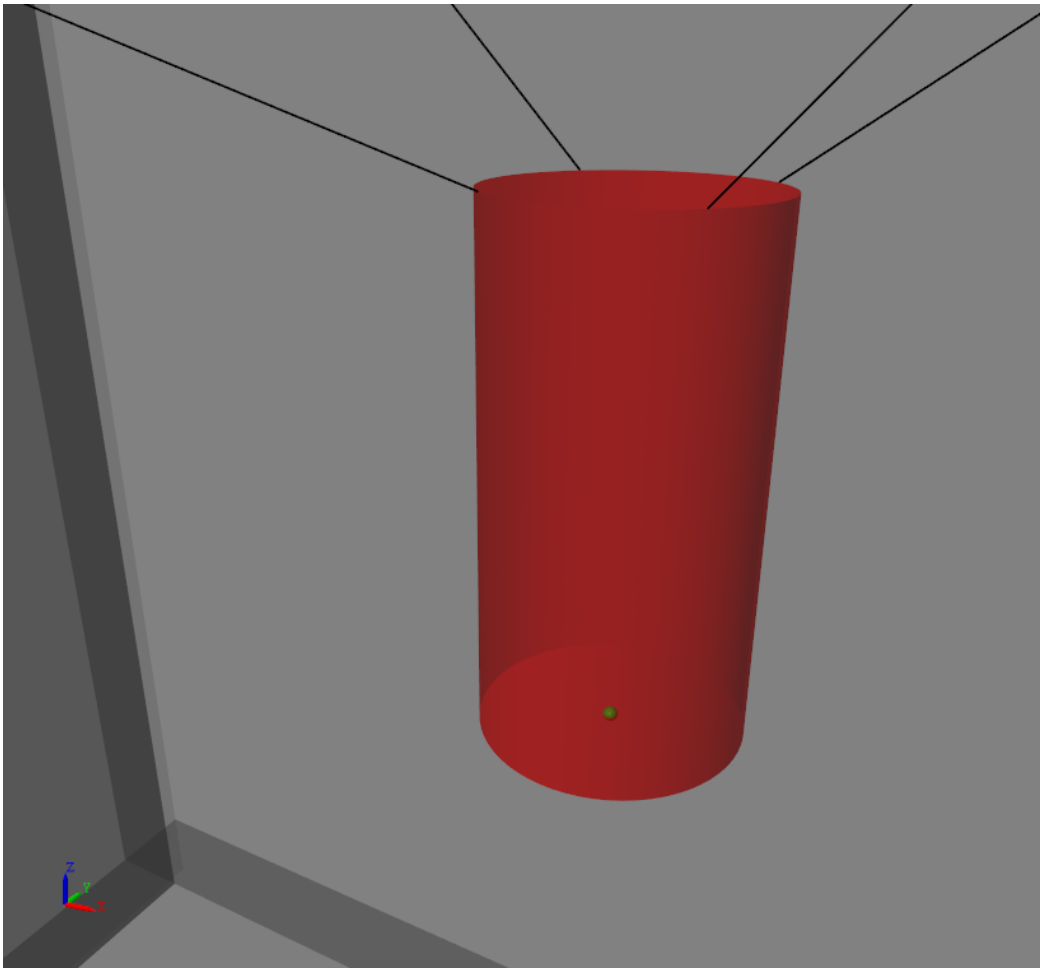


Figure 3.6: The modeled end efector with wires connected at the anchor points

## 3.3 Trajectory generation

To simplify the trajectory generation process as much as possible and to be able to easily create different trajectories, a structured approach was needed to both generate and implement the trajectory data in Simulink and REXYGEN. The following structure of the trajectory data was chosen

| | | | | | |
|---|---|---|---|---|---|
| 0 | $X_0$ | $Y_0$ | $Z_0$ | $v_0$ | $SoE_0$ |
| $t_1$ | $X_1$ | $Y_1$ | $Z_1$ | $v_1$ | $SoE_1$ |
| $t_2$ | $X_2$ | $Y_2$ | $Z_2$ | $v_2$ | $SoE_2$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $t_N$ | $X_N$ | $Y_N$ | $Z_N$ | $v_N$ | $SoE_N$ |

Table 3.1: The structure of trajectory points

where $t_0 - t_N$ represents the time [s], when the efector is required to arrive at this point, $X_0/Y_0/Z_0 - X_N/Y_N/Z_N$ represents the XYZ location [m] of the point, $v_0 - v_N$ represents the maximal velocity (number of pulses per second) of the stepper motors while approaching this point and finally $SoE_0 - SoE_N$ represents the state of the electromagnet used as the end-efector (0 - off, 1 - on).

Using this structure, it is possible to enter the required points of interest of the trajectory. To properly simulate the movement, a trajectory interpolator was created. The idea was to smooth out the movement between the given trajectory points in a way that the end efector stops in each of the provided points and to have a smooth transition. To achieve this, a cubic interpolation was used between the points with the limiting factor that the velocity in each of the points was equal to zero.

Any of the trajectory segments restricted by two consecutive points of interest $P(t_1) = \{X(t_1); Y(t_1); Z(t_1)\}$ and $P(t_2) = \{X(t_2); Y(t_2); Z(t_2)\}$ is given by a cubical polynomial:

$$X(t) = a_3^X \cdot t^3 + a_2^X \cdot t^2 + a_1^X \cdot t + a_0^X \tag{3.5}$$

$$Y(t) = a_3^Y \cdot t^3 + a_2^Y \cdot t^2 + a_1^Y \cdot t + a_0^Y \tag{3.6}$$

$$Z(t) = a_3^Z \cdot t^3 + a_2^Z \cdot t^2 + a_1^Z \cdot t + a_0^Z \tag{3.7}$$

First let's calculate the coefficients $a_3^X$, $a_2^X$, $a_1^X$ and $a_0^X$ of the polynomial in the X direction. If we differentiate both sides of the equation (3.5), we get

$$X'(t) = 3 \cdot a_3^X \cdot t^2 + 2 \cdot a_2^X \cdot t + a_1^X \tag{3.8}$$

Both (3.5) and (3.8) stated above have to be satisfied in both points of interest $P(t_1)$ and $P(t_2)$. Given that the velocity of the efector in both points of interest $P(t_1)$ and $P(t_2)$ is required to be equal to zero, we get the following system of 4 equations.

$$\begin{aligned} X(t_1) &= a_3^X \cdot t_1^3 + a_2^X \cdot t_1^2 + a_1^X \cdot t_1 + a_0^X \\ X(t_2) &= a_3^X \cdot t_2^3 + a_2^X \cdot t_2^2 + a_1^X \cdot t_2 + a_0^X \\ 0 &= 3 \cdot a_3^X \cdot t_1^2 + 2 \cdot a_2^X \cdot t_1 + a_1^X \\ 0 &= 3 \cdot a_3^X \cdot t_2^2 + 2 \cdot a_2^X \cdot t_2 + a_1^X \end{aligned} \tag{3.9}$$

$$\begin{bmatrix} X(t_1) \\ X(t_2) \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ 3 \cdot t_1^2 & 2 \cdot t_1 & 1 & 0 \\ 3 \cdot t_2^2 & 2 \cdot t_2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_3^X \\ a_2^X \\ a_1^X \\ a_0^X \end{bmatrix} \tag{3.10}$$

This system of equation can be solved by simply using the inverse of a matrix (given that the inverse exists).

$$\begin{bmatrix} a_3^X \\ a_2^X \\ a_1^X \\ a_0^X \end{bmatrix} = \begin{bmatrix} t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ 3 \cdot t_1^2 & 2 \cdot t_1 & 1 & 0 \\ 3 \cdot t_2^2 & 2 \cdot t_2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} X(t_1) \\ X(t_2) \\ 0 \\ 0 \end{bmatrix} \tag{3.11}$$

This solution can be used as well for the polynomials in the Y and Z direction as described below.

$$\begin{bmatrix} a_3^Y \\ a_2^Y \\ a_1^Y \\ a_0^Y \end{bmatrix} = \begin{bmatrix} t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ 3 \cdot t_1^2 & 2 \cdot t_1 & 1 & 0 \\ 3 \cdot t_2^2 & 2 \cdot t_2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} Y(t_1) \\ Y(t_2) \\ 0 \\ 0 \end{bmatrix} \tag{3.12}$$

$$\begin{bmatrix} a_3^Z \\ a_2^Z \\ a_1^Z \\ a_0^Z \end{bmatrix} = \begin{bmatrix} t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ 3 \cdot t_1^2 & 2 \cdot t_1 & 1 & 0 \\ 3 \cdot t_2^2 & 2 \cdot t_2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} Z(t_1) \\ Z(t_2) \\ 0 \\ 0 \end{bmatrix} \tag{3.13}$$

The calculated coefficients from (3.11), (3.12) and (3.13) can then be used together with (3.5), (3.6), (3.7) respectively to describe the trajectory in between the two points of interest $P(t_1)$ and $P(t_2)$ used to calculate the coefficients.

### 3.3.1 Example Trajectory

As an example of the described cubic interpolation, the following points of interest were used in an example trajectory.

| t[s] | X[m] | Y[m] | Z[m] |
|---|---|---|---|
| 0 | 0 | 0 | 0.25 |
| 1 | 0.125 | 0 | 0.25 |
| 2 | 0.250 | 0.125 | 0.25 |
| 3 | 0 | 0 | 0 |

Table 3.2: Selected points of interest of the example trajectory

After calculating the coefficients for each of the segments of the trajectory between the given points of interest, we can plot the whole trajectory. It is apparent that at the given times, the object/efector is present at the correct location and the derivative of it's position (velocity) is equal to zero.

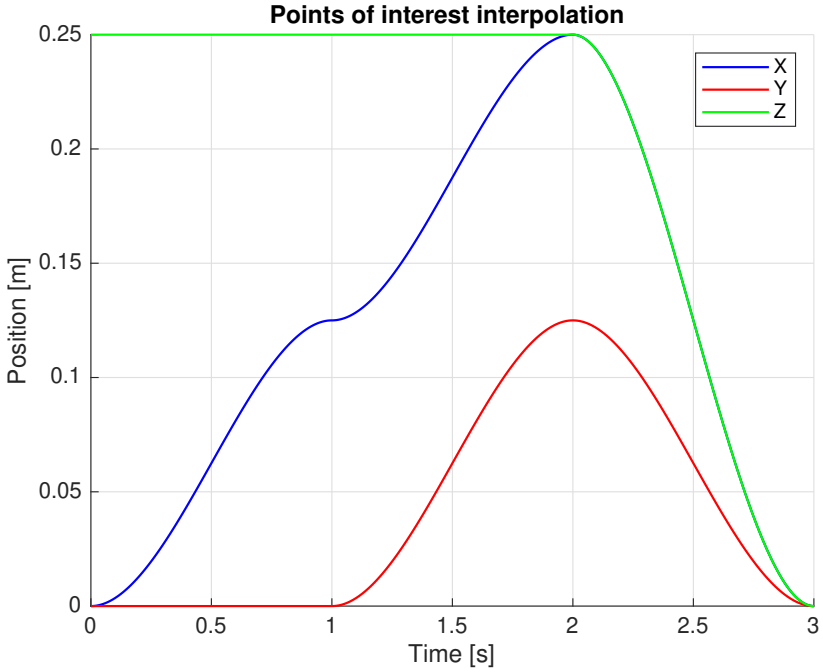



Figure 3.7: Interpolation of an example trajectory given by points of interest defined in 3.2

## 3.4 Model in the loop

As the next step, the inverse kinematic task (2.2) was implemented as a Matlab function and used together with the described model in a MIL simulation. As an input, first manual XYZ position sliders were used to move the end efector in real time. After that a simple circle trajectory at a constant height was created to further test the functionality of both the model and the implemented inverse kinematic task.

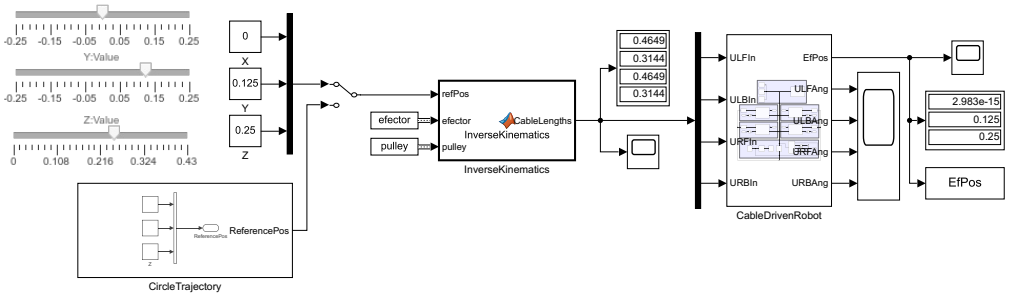



Figure 3.8: Control interface of the cable robot simulation with the inverse kinematic task and trajectory generation

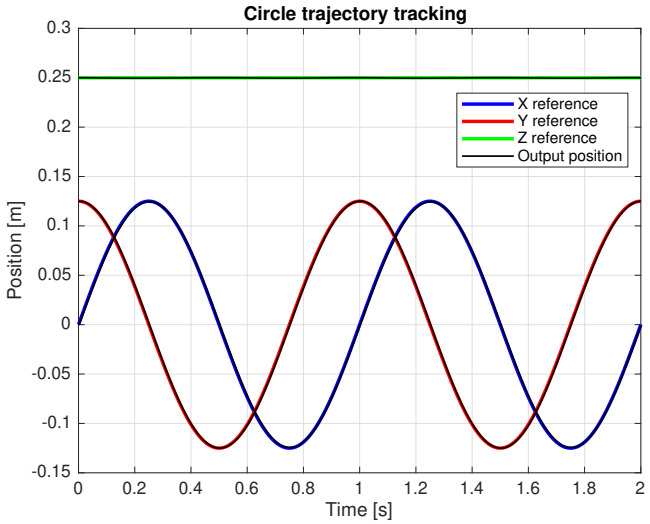



Figure 3.9: Comparison of the desired trajectory and model output for the circle trajectory

As seen in (3.9), the model tracks the desired input trajectory in all axes even for a multi-axis movement.

# Chapter 4

# Experimental Physical Model

To test the implemented functionality in practice and to check any differences in the real life implementation, it was decided that an experimental physical model would be created. The goal of this experimental model was to learn about possible ways to implement the electrical and mechanical interface into the final product and connect them to the software to test the inverse kinematic task with trajectory generation.

## 4.1 Electrical Interface

To implement the software, Rexygen studio on RaspberryPi + Monarco Hat was used. For spool actuators, stepper motors were chosen due to their higher accuracy in position control. Because of the relatively high sample period that could be reached on the target platform, the decision was made to use a communication protocol to control the position of the stepper motors. Because of this, four JSS57P2N closed loop stepper motors with a RS232 Modbus RTU control interface were chosen. To connect all 4 stepper motors to the RaspberryPi, 4 USB-RS232 converters together with a USB hub were used. Power supply of 24V/15A was enough to properly power up all of the electronics. In order to have an actuated end efector, a 50N 24V electromagnet was screwed to the tip of the end efector. This electromagnet was connected through a relay to the AOUT1 output of the MonarcoHat. Because of this, an on-demand actuation was achieved from the Rexygen studio environment.

In the picture below (4.1), we can see the top view of the cable robot. The respective components are marked with colored circles. The JSS57P2N stepper motors are marked as red, the RS232-USB converters are marked as green, the RaspberryPi + MonarcoHat together with the USB hub are marked as light blue, the relay connected to the end efector is marked as yellow and the power supply is marked are purple.

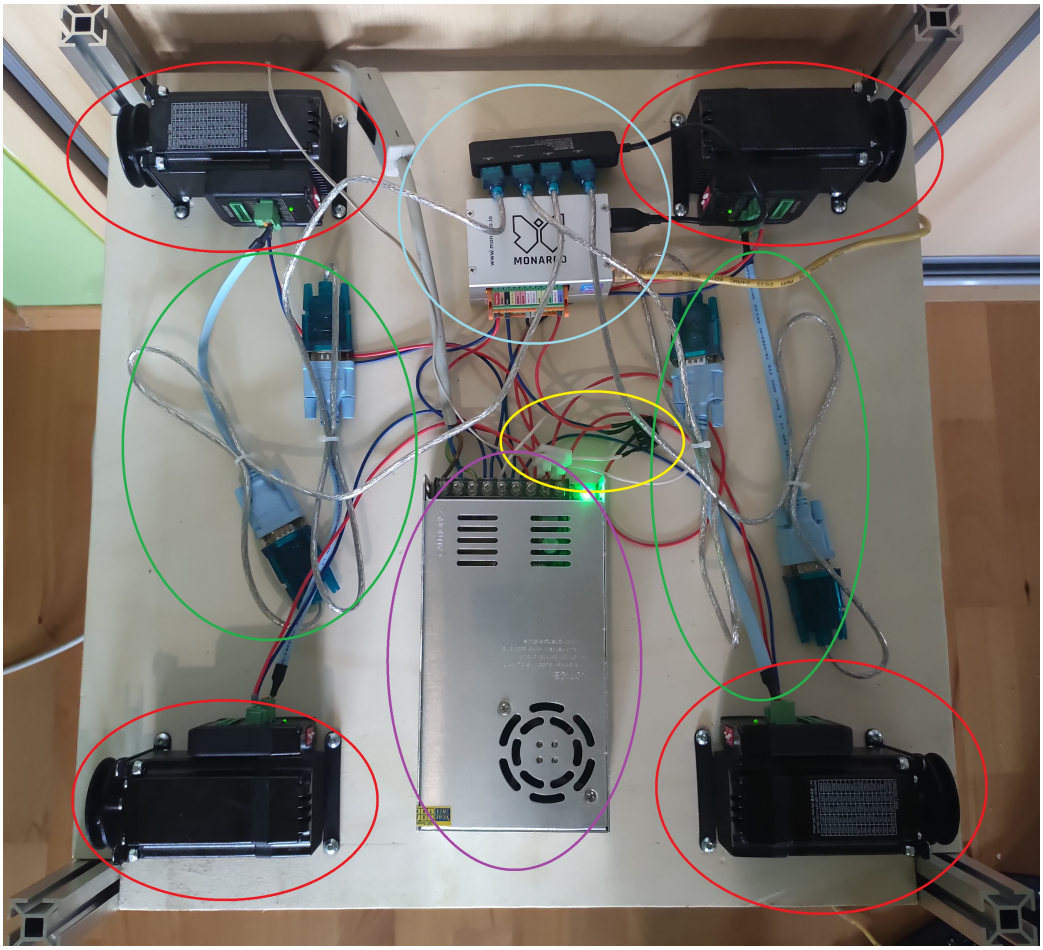


Figure 4.1: Electronics of the Cable robot (top view). Red - stepper motors. Green - RS232-USB converters. Light Blue - RaspberryPi + MonarcoHat + USB hub. Yellow - efector relay. Purple - Power Supply

From the instruction manual provided by the JSS stepper motor dealer, the following list of parameters was exported. It shows all of the internal (closed loop PID controller parameters, filtering times,...) and external (desired position, maximal velocity/acceleration,...) parameters that are possible to be set and read using the RS232 interface.

| Modbus Item Address | Parameter Name | Used Value |
|---|---|---|
| 0 | Model number | 57 |
| 1 | Open/Closed loop | 1 |
| 2 | Motor Type | 0 |
| 3 | Current loop Kp | 1000 |
| 4 | Current loop Ki | 200 |
| 5 | Position loop Kp | 300 |
| 6 | Speed loop Kp | 400 |
| 7 | Speed loop Ki | 80 |
| 8 | Default number of pulses per rev | 400 |
| 9 | Encoder resolution | 4000 |
| 10 | Tracking error alarm threshold | 4000 |
| 11 | Open loop holding current | 30 |
| 12 | Closed loop holding current | 60 |
| 13 | Pulse command filtering time | 60 |
| 14 | Enable level polarity | 1 |
| 15 | Fault output level polarity | 0 |
| 16 | Pulse input mode | 0 |
| 17 | Pulse effective edge | 0 |
| 18 | PEND output function | 0 |
| 19 | PEND output level polarity | 0 |
| 20 | Low acceleration | * |
| 21 | High acceleration | 0 |
| 22 | Low deceleration | * |
| 23 | High deceleration | 0 |
| 24 | Low maximum speed | * |
| 25 | High maximum speed | 0 |
| 26 | Target pulse count | * |
| 27 | Total number of pulses | 0 |
| 28 | Motion control instruction | * |
| 29 | Position Mode | 1 |
| 30 | Absolute position is low | * |
| 31 | Absolute position | 0 |
| 32 | Internal pulse state | 1 |
| 33 | Save parameters | 0 |
| 34 | Factory reset | 0 |

Table 4.1: List of parameters of the JSS57P2N stepper motor (* - Parameter changed from the control system)

To successfully control the JSS stepper motors over the RS232 interface was problematic and was caused mainly by the scarce retailer specification. I found a "Protuner" software that was working with similar stepper motors and tried to connect it to one of the JSS stepper motors using a COM interface on a Windows PC. Even though most of the parameters are not well described in the specification, I was able to recognize the most important parameters of the stepper motor for our application. Those important parameters include

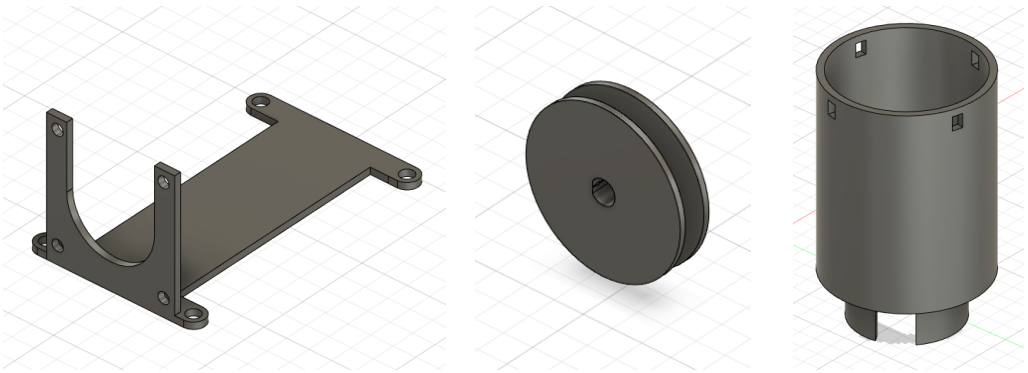
- Low acceleration (20), Low deceleration (22) and Low maximum speed (24) as saturation limits for maximal allowed acceleration, deceleration and speed respectively
- Target pulse count (26) as the desired absolute position of the stepper motor
- Motion control instruction (28)
  - 0 - Standby
  - 1 - Fixed length motion
  - 2 - Continuous motion
  - 3 - Deceleration stop
  - 4 - Stop immediately
- Absolute position is low (30) to set the current absolute position to a new value (homing purposes)

Note: The parameter "High maximum speed (25)" was is not being manipulated in the Protuner software and so here the same parameter "Low maximum speed (24)" is used instead. Unfortunately due to the lack of description in the documentation, the reason for this is unknown. The same applies to the acceleration and deceleration limits as well.

## 4.2 Mechanical Interface

To properly construct the model, Autodesk Fusion 360 was used to model and visualize each of the end components. For the main body, 30x30 mm aluminium profiles were precut and used (8x 500mm + 4x 650mm). Plywood (8mm thickness) was used at the top and bottom as a base for the electronics. Small holes were made in the top plywood layer to guide the robot cables to the end efector in order to simplify the inverse kinematics calculation (as discussed and modelled in previous chapters). The horizontal workspace remained equal to the Simulink model (500x500 mm).

In order to fixate the position of the stepper motors, 4 holders were 3D-printed and used to mount the motors. Furthermore, 4 spools with the diameter of 30 mm were also 3D-printed and used to coil 0.38mm wire. This wire is both strong and non-flexible to result in as much length accuracy as possible while also being easy to be coiled and retaining it's shape. Finally an end efector was 3D printed as well to house the electromagnet. It has four holes at the top, each one for a carbine tied to the wire. In the middle of the end efector, there is space for a 500g weight. This helps to increase downward force with gravity, making the efector less likely to vibrate from the motion. All of those modeled parts can be seen below (4.2).



(a) Motor Holder (b) Spool (c) End Efector

Figure 4.2: All 3D printed parts for the cable robot

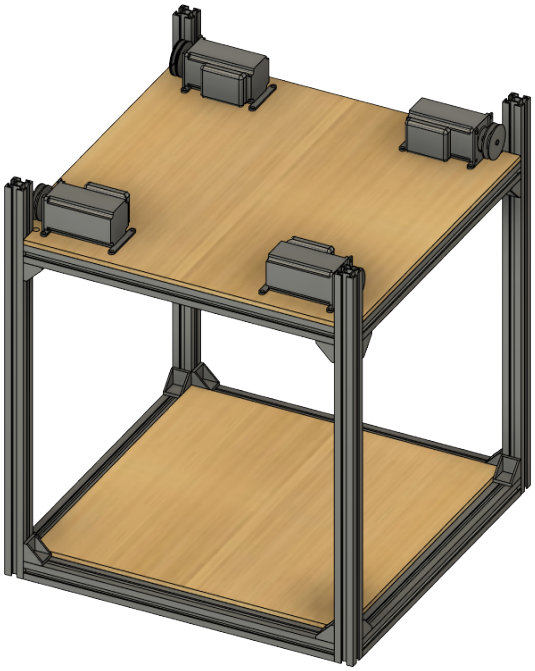


Figure 4.3: Experimental model visualization from Autodesk Fusion 360

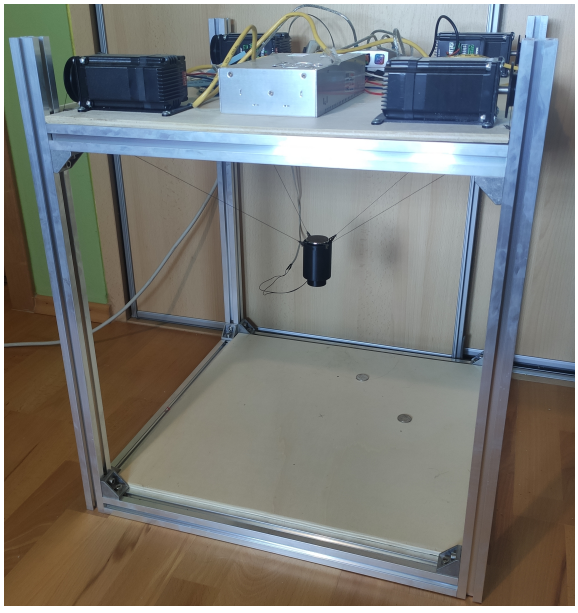


Figure 4.4: Real-life construction of the experimental model.

## 4.3 Software

REXYGEN studio 2.50.99.10936 was used to program the RaspberryPi + Monarco HAT combination. The IP address of the used RaspberryPi was set to 192.168.0.202. Only one model task was used with the sample tick time of 5 ms. The basic MonarcoDrv module was used to actuate the electromagnet at the end efector. To communicate with the JSS57P2N stepper motors and control their desired position, Modbus RTU communication protocol was used over an RS232 hardware interface.

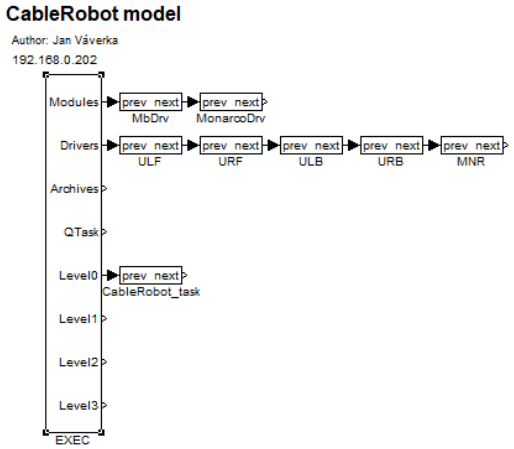



Figure 4.5: The executive file of the Rexygen project.

In order to properly distinguish each of the motors, REXYGEN's MbDrv module was connected and used to program a Master/Slave Modbus communication for each of the motors, with their respective serial address of the USB port connected. Those addresses are described in the table below (4.2).

| | |
|---|---|
| ULF port | /dev/ttyUSB0 |
| URF port | /dev/ttyUSB1 |
| ULB port | /dev/ttyUSB2 |
| URB port | /dev/ttyUSB3 |

Table 4.2: USB ports used to connect the stepper motors and to establish the connection

To correctly initialize the USB ports during the RaspberryPi boot process, it is mandatory to connect all of the USB-RS232 converters one by one AFTER the RaspberryPi is booted up in the order given in 4.2.

That way, all of the converters are assigned to the correct port, which allows the individual motors to be linked correctly to their respective signal in the REXYGEN environment.

$$ULF \rightarrow URF \rightarrow ULB \rightarrow URB$$

This address is also assigned in the Modbus driver of each of the motors. The correct Modbus settings for the ULF stepper motor can be seen below, along with the defined messages that are being sent and received over the communication. The specific messages refer to the parameters defined by the stepper motor retailer in (4.1).

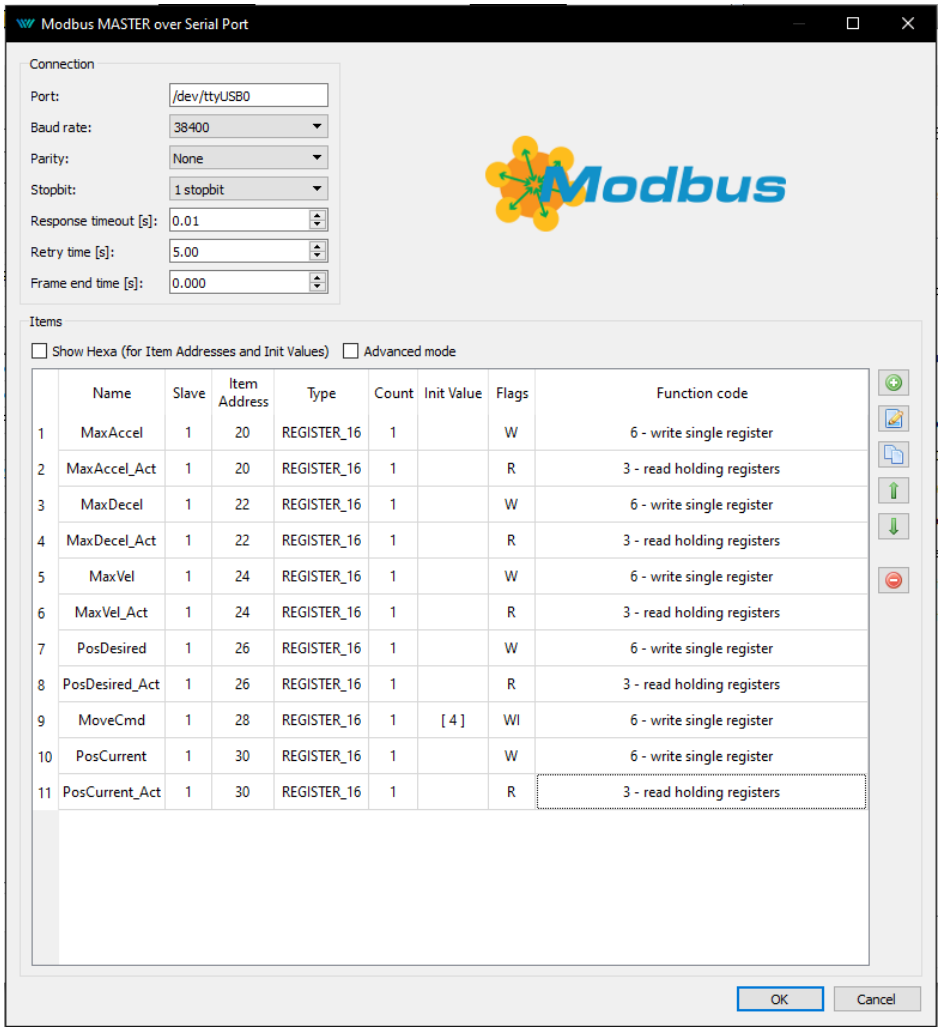



Figure 4.6: Modbus driver settings of the ULF motor

Each message is either readable or writable relative to the communication. In the case that one register is able to be written to and read from at the same time, two messages are created to split the functionality and a $*\_Act$ suffix is added to the name of the item which reads the register. In the model task, flags referencing to drivers are marked as yellow.

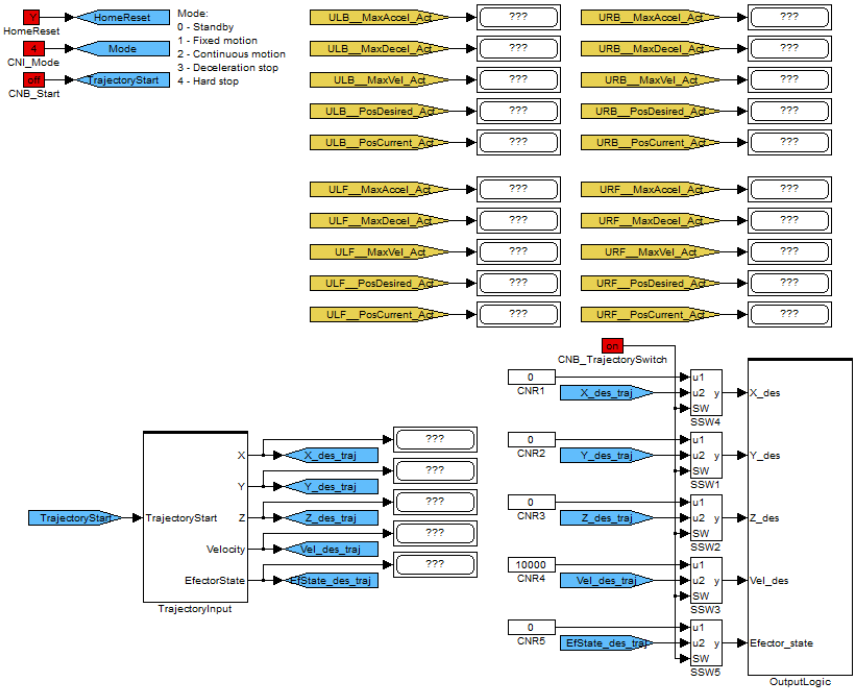



Figure 4.7: The main task of the Rexygen project. Driver flags are marked as yellow. User control variables are marked as red.

The task can be split into several parts which be described below. We already talked about the driver flags which read the required position (in pulses), current position and current velocity/acceleration limits for each motor. Following that, we have a TrajectoryInput subsystem, which imports the required trajectory (also described in the previous chapter). Another subsystem called OutputLogic is used to calculate the inverse kinematic task and send the desired position set points to their respective motors. It is also used to actuate the efector's electromagnet and set other motor parameters.

Another functionality on the main level is the user control interface, which is created by four red constants that allow the user to set the current position of all the motors ($HomeReset$), activate/stop the motors ($CNI\_Mode$), start the trajectory motion ($CNB\_Start$) or select to disable the trajectory and only move to a constant XYZ position ($CNB\_TrajectorySwitch$).

### 4.3.1 Subsystem OutputLogic

In this subsystem, the desired position in the XYZ coordinates is saturated to the physical model dimensions (due to safety). After that, the saturated position is used to calculate the lengths of the respective wires/tendons of the cable robot. Since we know exactly the radius of the 3D printed spools ($R = 15mm$) and the number of pulses per revolution of the stepper motors (3200), we can calculate the required absolute pulse position of the motors simply by using gain blocks.

$$GAIN\_Len2Rot : k = \frac{1}{2 \cdot \pi \cdot R} \sim 0.0106 \tag{4.1}$$

$$GAIN\_Rot2Pulse : k = 3200 \tag{4.2}$$

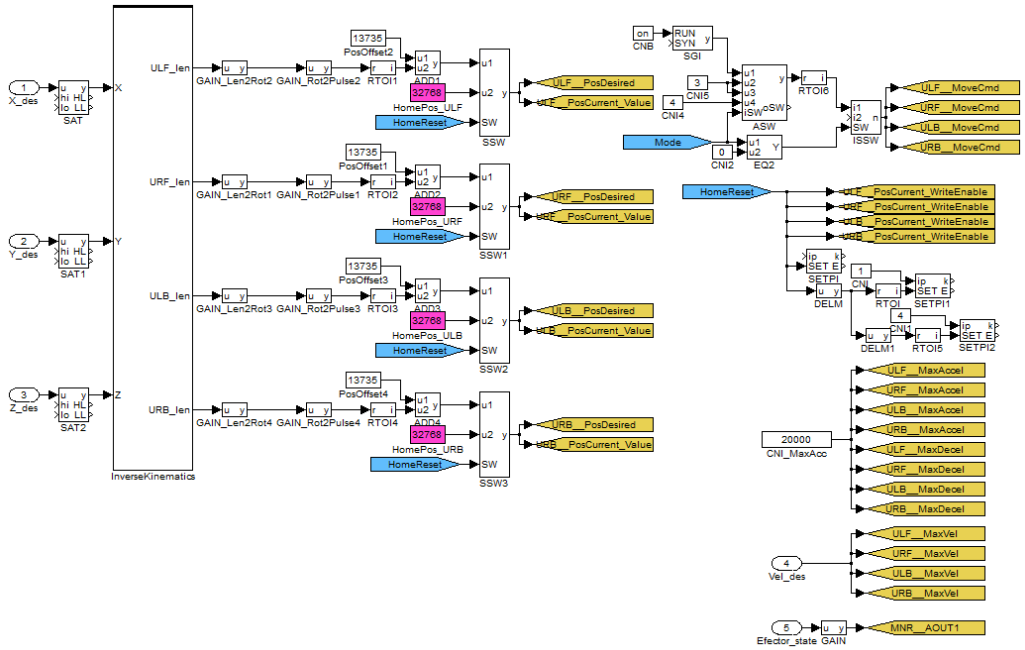



Figure 4.8: The OutputLogic subsystem of the main task. Driver flags are marked as yellow.

Because the Modbus message TargetPulseCount (26) is defined as an unsigned integer by the retailer, a constant offset is then added to keep the current position away from zero at all times. This prevents the the desired position from overflowing and the efector to go in an unwanted direction. The value of this offset is chosen in such a way that the position set point of the robot at the homing point ([0;0;0]) is equal to the middle of the 16-bit

message range (32768). With the currently used parameters in the inverse kinematic task, an offset of 13735 pulses is needed to achieve this. The following switch is used to set the correct position during homing. Since our homing point is directly at [0;0;0] (middle of the workspace at the ground level), then the values of 32768 pulses is used to set the position.

To correctly home the motor, it is needed to manually move the end efector to the home position at the ground level in the middle of the workspace before turning the power on. After making sure that all of the wires are correctly tensed, pressing the HomeReset button will send the homing command (this should be done directly after every startup to avoid any unwanted movement).

After a homing command is received, a sequence starts that stops the movement in case the efector is not idle and starts the movement for on tenth of a second to correctly set the current position to the motor. If the motor's command is not set to Fixed Motion (1) right after changing the current position, then the motor doesn't save the position into memory and the homing will not be successful.

Apart from this functionality, other parameter setting is done in the subsystem as well, including the acceleration/deceleration settings, velocity setting and efector's electromagnet actuation.

### 4.3.2 Subsystem InverseKinematics

To correctly interpret the inverse kinematic task, the InverseKinematics subsystem was created. As stated in an earlier chapter (2.2), the inverse kinematic task is given by the following expression

$$l_i = \|\mathbf{b_i} - \mathbf{r} - \mathbf{R} \cdot \mathbf{p_i}\|_2 \tag{4.3}$$

With $\mathbf{b_i}$ representing the XYZ eye position of the i-th cable, $\mathbf{r}$ representing the desired XYZ position of the end efector and $\mathbf{p_i}$ representing the anchor XYZ position in relation to the end efector tip. Since in this project we only use four cables and commanding the rotation of the end efector is not possible, rotational axis $\mathbf{R}$ was set to an identity matrix and is not used in this task. The rest of the parameters are defined in the subsystem mask (4.3) and used in CNR blocks to calculate the correct cable lengths.

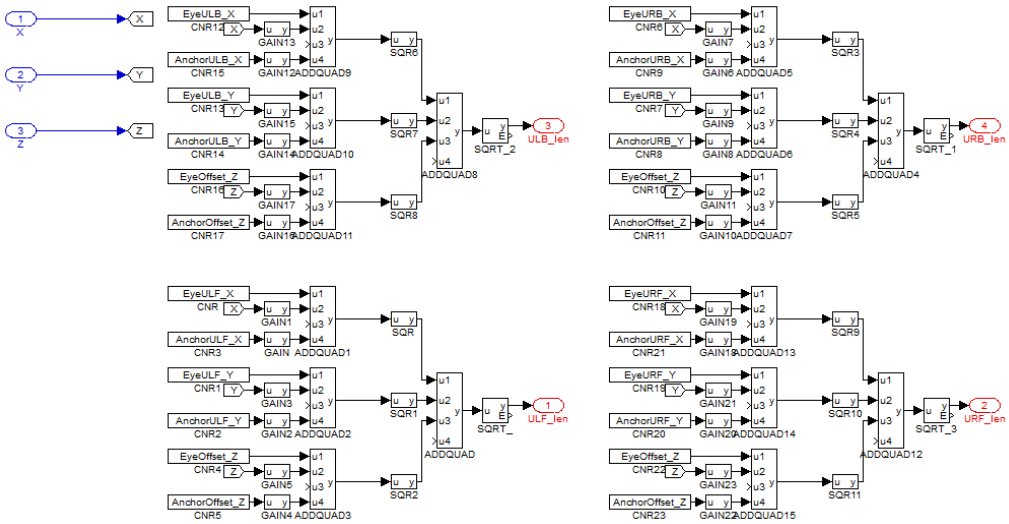



Figure 4.9: The InverseKinematics subsystem of the main task.

| Parameter name | Value |
|---|---|
| AnchorOffset_Z | 68 |
| AnchorULF_X | -15 |
| AnchorULF_Y | -15 |
| AnchorURF_X | 15 |
| AnchorURF_Y | -15 |
| AnchorULB_X | -15 |
| AnchorULB_Y | 15 |
| AnchorURB_X | 15 |
| AnchorURB_Y | 15 |
| EyeOffset_Z | 522 |
| EyeULF_X | -247.5 |
| EyeULF_Y | -247.5 |
| EyeURF_X | 247.5 |
| EyeURF_Y | -247.5 |
| EyeULB_X | -247.5 |
| EyeULB_Y | 247.5 |
| EyeURB_X | 247.5 |
| EyeURB_Y | 247.5 |

Table 4.3: Parameters used in the inverse kinematic task

### 4.3.3 Subsystem TrajInput

This subsystem loads the trajectory .csv file previously generated in the Matlab environment and takes care of selecting the correct point for any time of the simulation.

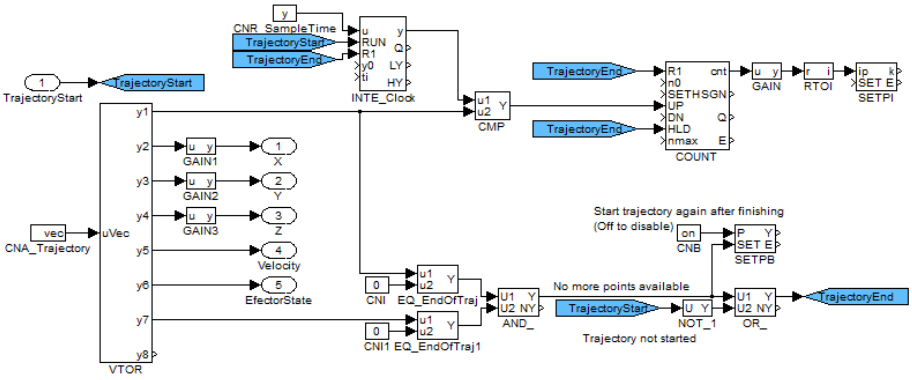



Figure 4.10: The TrajInput subsystem of the main task.

The .csv file has the same structure as the one used in the Simulink simulation. However since the position is already regulated by the motor retailer, only the raw non-interpolated version is used. The trajectory is loaded by a VTOR block. The VTOR offset parameter is then increased every time we need to select the next point of interest of the trajectory. To select the next point, the trajectory index/offset must be increased by 6, since that's the number of parameters used for each point, which can be seen in the table below.

| | | | | | |
|---|---|---|---|---|---|
| 0 | $X_0$ | $Y_0$ | $Z_0$ | $v_0$ | $SoE_0$ |
| $t_1$ | $X_1$ | $Y_1$ | $Z_1$ | $v_1$ | $SoE_1$ |
| $t_2$ | $X_2$ | $Y_2$ | $Z_2$ | $v_2$ | $SoE_2$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $t_N$ | $X_N$ | $Y_N$ | $Z_N$ | $v_N$ | $SoE_N$ |

Table 4.4: The structure of trajectory points

To correctly track the time, an integrator is used as an internal clock. Once the internal clock reaches the time of the next point, the offset is increased and with it the next point. Once the time of the next point is at 0, it is clear that we are at the end of the trajectory,the movement can be stopped and the offset can be reset. It is also possible to continue the trajectory motion from the start after finishing the trajectory, which results in a periodic motion.

# Chapter 5

# Model comparison

To compare the model created in the Matlab/Simulink environment and it's physical real-world representation, a complete trajectory was created. It consists of 25 points of interest and is 32 seconds long. With this trajectory, the cable robot is supposed to handle two plate-like ferromagnetic objects between two places. The points of interest are defined in (5.1). Those points of interest were interpolated as described in 3.3 and used as a modified trajectory for the Simulink model.

Since the JSS stepper motors used in the physical model have their own position controller from the retailer, only the raw points of interest (without any interpolation) are used to control the position. This means that the movement between two specific points of interest might not be exactly the same. In the physical model, this behaviour has to be offset by the velocity and acceleration/deceleration settings of each motor. Another option would be to plan the points of interest times in such a way that the stepper motors are fast enough to reach the next point as closely as possible to avoid any idle time. In this example, only constant velocity and acceleration/deceleration setting was used for each point to show the obvious difference between the trajectories.

In the figure (5.1), the actual cable lengths for all cables can be observed (red), together with the point's of interest set points (blue) and the interpolated trajectory for the Simulink model (magenta). It is clear, that the physical model is faster to reach the desired set point and remains idle until the next set point comes. This behaviour can be also seen in a video comparing the Simulink simulation and the Rexygen model measurement.

`https://www.youtube.com/watch?v=KF2o1H832cc`

| t[s] | X[m] | Y[m] | Z[m] | $v_{max}$[pulse/s] | SoE |
|---|---|---|---|---|---|
| 0.0 | 0 | 0 | 0.250 | 5000 | 0 |
| 2.0 | 0.125 | 0 | 0.005 | 20000 | 0 |
| 2.1 | 0.125 | 0 | 0.005 | 20000 | 1 |
| 4.0 | 0 | 0 | 0.150 | 20000 | 1 |
| 6.0 | -0.125 | 0 | 0.005 | 20000 | 1 |
| 6.1 | -0.125 | 0 | 0.005 | 20000 | 0 |
| 8.0 | 0 | 0 | 0.150 | 20000 | 0 |
| 10.0 | 0 | 0.125 | 0.005 | 20000 | 0 |
| 10.1 | 0 | 0.125 | 0.005 | 20000 | 1 |
| 12.0 | 0 | 0 | 0.150 | 20000 | 1 |
| 14.0 | 0 | -0.125 | 0.005 | 20000 | 1 |
| 14.1 | 0 | -0.125 | 0.005 | 20000 | 0 |
| 16.0 | 0 | 0 | 0.150 | 20000 | 0 |
| 18.0 | -0.125 | 0 | 0.005 | 20000 | 0 |
| 18.1 | -0.125 | 0 | 0.005 | 20000 | 1 |
| 20.0 | 0 | 0 | 0.105 | 20000 | 1 |
| 22.0 | 0.125 | 0 | 0.005 | 20000 | 1 |
| 22.1 | 0.125 | 0 | 0.005 | 20000 | 0 |
| 24.0 | 0 | 0 | 0.150 | 20000 | 0 |
| 26.0 | 0 | -0.125 | 0.005 | 20000 | 0 |
| 26.1 | 0 | -0.125 | 0.005 | 20000 | 1 |
| 28.0 | 0 | 0 | 0.150 | 20000 | 1 |
| 30.0 | 0 | 0.125 | 0.005 | 20000 | 1 |
| 30.1 | 0 | 0.125 | 0.005 | 20000 | 0 |
| 32.0 | 0 | 0 | 0.150 | 20000 | 0 |

Table 5.1: Points of interest of the cyclic pick&place trajectory

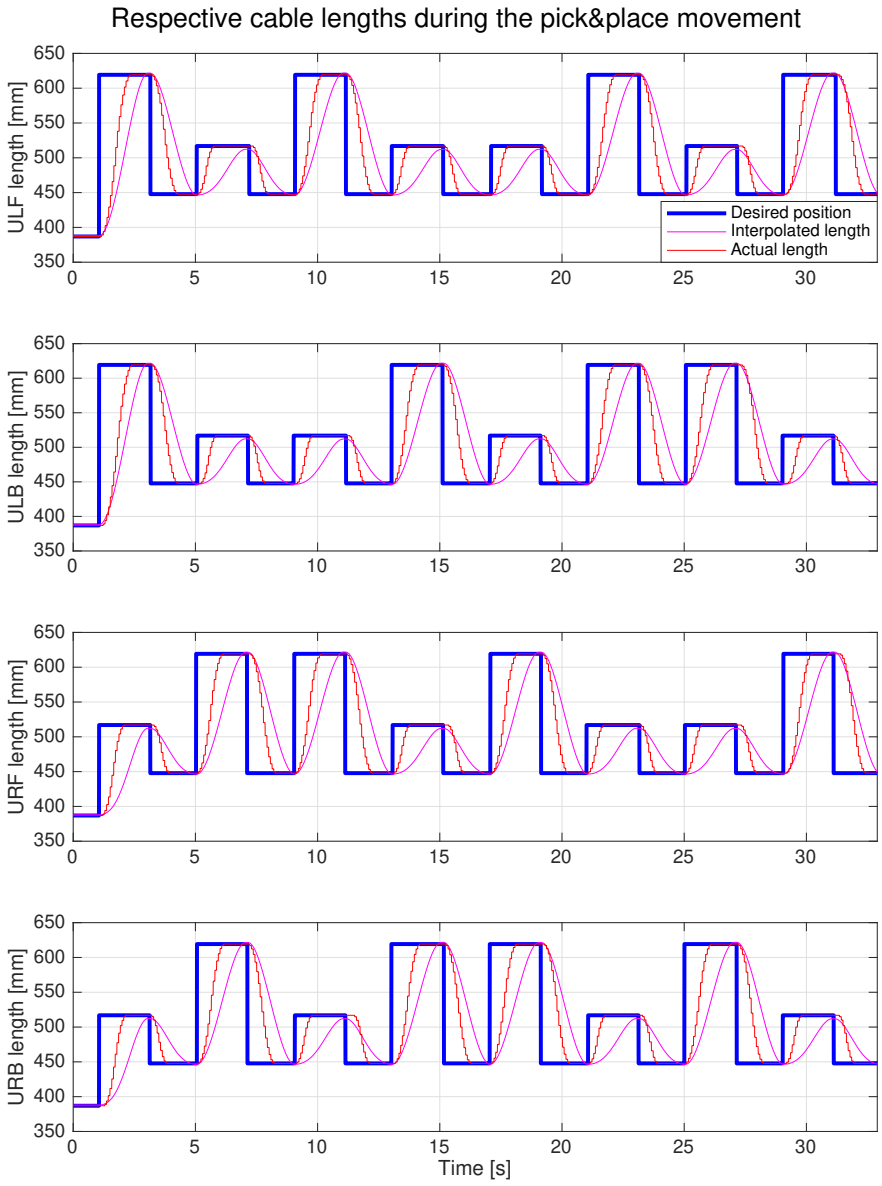



Figure 5.1: Time plot showing the desired cable length set points (blue), interpolated lengths (magenta) and actual movement measured from Rexygen (red)

# Chapter 6

# Conclusion

The aim of this thesis was to study the theory of cable-driven parallel manipulators and the possibilities to construct mathematical models of such manipulators. This includes the analysis of the workspace, limitations, kinematic tasks and also trajectory generation.

To summarize, parallel manipulators in general are more sturdy and safe, however in general they have a smaller workspace that is limited by their construction (e.g Steward platform). This limitation is partially lifted with cable-driver manipulators, since their actuators consist of an actuated spools that reels in/out the cables. Out of the two kinematic tasks, the inverse kinematic task is computationally easier with only one solution computed as an euclidean norm. The forward kinematic task on the other hand doesn't have an analytical solution and has to be solved numerically by finding the intersection of $n$-spheres, where $n$ is the number of cables. After solving the inverse kinematic task, the issue of trajectory generation is solved as well, since we can transform the coordinates in the XYZ coordinate system into the manipulator coordinates (in this case the cable lengths), which can then be used to control the manipulator. To create more complex trajectories, a csv format structure was created together with an interpolating algorithm.

The simulation model was created in the Matlab/Simulink environment using the Simscape Multibody library. This allowed me to accurately simulate the movement of each cable together with the spool actuation and to test the model together with the trajectory generator in a MiL simulation.

It was also decided to create a physical real-world model using Rexygen studio. For this purpose, a Fusion360 project was created to plan the mechanical construction. The main body was constructed from aluminium 30x30 profiles with the base made of plywood. Several components were 3D-printed using the Ender 3 printer to house the electronics and connect them to the mechanical construction. As for the electrical interface, RaspberryPi + MonarcoHat was used and programmed to control 4 JSS57P2N closed loop stepper motors using ModbusRTU over an RS232 interface. A strong electromagnet was used together with a relay to act as an end efector. There was a difference in the movement between the Simulink model and mechanical model caused by the internal PID controller of the stepper motors. However this behaviour could be offset by limiting the internal maximal velocity of the motors.

Generally the mechanical construction was sufficient and precise enough. However the Modbus communication is somewhat fragile and slow. To improve the model, another form of control would have to be implemented. Ideally a stepper motor without any internal controller/driver. That way a PFM driver and manual control loop would have to be implemented to achieve smoother movement.

Another upgrade of the model would of course be to add degrees of freedom by adding 4 more actuated spools that connect to the end efector from the bottom. This would add the possibility to control the rotation of the end efector and allow for a faster movement. However for pick&place applications, this is not always possible.

# Bibliography

[1] Bruckmann, Tobias & Mikelsons, Lars & Brandt, Thorsten & Hiller, Manfred & Schramm, Dieter. (2008). *Wire Robots Part I: Kinematics, Analysis & Design.*

[2] Bruckmann, Tobias & Mikelsons, Lars & Brandt, Thorsten & Hiller, Manfred & Schramm, Dieter. (2008). *Wire Robots Part II: Dynamics, Control & Application.*

[3] Morris, Melissa & Shoham, Moshe. (2009). *Applications and Theoretical Issues of Cable-Driven Robots.*

[4] Gosselin, Clément. (2013). *Cable-driven parallel mechanisms: state of the art and perspectives.*

[5] Švejda, Martin. (2016). *Optimalizace Robotických Architektur.*

[6] Sciavicco, Lorenzo. (2005). *Modelling and Control of Robot Manipulators.*

[7] Abdolshah, Saeed. (2016). *Trajectory Planning and Control of Cable-Driven Parallel Robots.*