Lund, Jørgen Meland
Sønderland, Henning
Vos, Jesper

3DOF Motion Platform For Educational Applications With Model Predictive Control

Bachelor's thesis in Electrical Engineering - Automation and Robotics
Supervisor: Coates, Erlend
May 2023

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

Lund, Jørgen Meland
Sønderland, Henning
Vos, Jesper

3DOF Motion Platform For Educational Applications With Model Predictive Control

Bachelor's thesis in Electrical Engineering - Automation and Robotics
Supervisor: Coates, Erlend
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

3DOF Motion Platform For Educational Applications With Model Predictive Control

Jørgen Meland Lund, Henning Sønderland & Jesper Vos

May 2023

BACHELOR'S THESIS

Department of ICT and Natural Sciences

Norwegian University of Science and Technology

Supervisor: Erlend Coates

Preface

This thesis is written by three students from the bachelor's degree programme in Electrical Engineering, study programme Automation and Robotics, NTNU Ålesund, Norway.

What intrigued us is the lack of good practical educational tools in the control theory field. Especially the lack of practical development tools for Model Predictive Control strategies. We were eager to explore the possibilities of making a versatile motion platform that allows other people to make their own controllers.

Acknowledgements

First and foremost, we would like to thank our supervisor, Erlend Coates, for providing us with the idea and support. This project would not have been completed without his guidance and feedback. We would also like to thank Anders Sætersmoen for providing us with parts and possible solutions to problems we encountered throughout the project. Last but not least, we are grateful for our friends and families whose support and encouragement ensured we were motivated throughout our project.

Abstract

The motion platform is an excellent tool for students to simulate real-world scenarios in a controlled environment for testing and experimenting. Model Predictive Control is seen as a more viable option in control methods due to recent improvements in algorithm development. MPC controls a system by calculating the appropriate control action by using measurements and a dynamical model of the system to predict future states. There is a lack of good practical educational tools to experiment with using this control strategy due to its computational requirements and complex nature. MPC, traditionally, is not commonly taught to undergraduates pursuing a bachelor's degree. It is often regarded as an advanced control technique and thus more frequently encountered at the master's level. However, incorporating MPC into a bachelor's program can introduce the topic to undergraduate students, familiarizing them and potentially sparking greater interest in it. In this thesis, a 3DOF motion platform that is both robust and user-friendly has been developed and built. This motion platform is developed to serve as an educational and experimental platform for students. It is developed to allow others to easily modify the design and implement their own controllers and systems. The platform is also equipped with a touch-screen to allow others to easily tune and change parameters for their controllers. All scripts are programmed in Python and are modular, such that each script can be replaced. To meet the requirement for computational power, a Khadas VIM3 single-board computer serves as the system for all software. This allows the MPC to easily control a ball on the top with minimal errors.

Sammendrag

En Bevegelsesplattform er et utmerket verktøy som studenter kan bruke til å simulere virkelige scenarioer i et kontrollert miljø til testing og eksperimentering. Model Predictive Control betraktes som et mer gjennomførbart alternativ innen kontrollmetoder på grunn av nylige forbedringer innen algoritmeutvikling. MPC styrer et system ved å beregne den optimale kontroll handlingen ved å bruke målinger og en dynamisk modell av systemet for å forutsi fremtidige tilstander. Det er en mangel på gode praktiske utdanningsverktøy for å eksperimentere med denne kontrollstrategien på grunn av dens beregningskrav og komplekse natur. Tradisjonelt sett blir MPC ikke vanligvis undervist til studenter som tar en bachelorgrad. Det blir ofte betraktet som en avansert kontrollteknikk og blir derfor oftere brukt på masternivå. Imidlertid kan bruk av MPC i et bachelorprogram introdusere emnet for studenter, gjøre dem kjent med det, og potensielt vekke større interesse for det. I denne hovedfagsoppgaven er det utviklet og bygget en 3DOF bevegelsesplattform som er både robust og brukervennlig. Denne bevegelsesplattformen er utviklet for å fungere som en utdannings- og eksperimentell plattform for studenter. Den er utviklet for å tillate andre å enkelt modifisere designet og implementere sine egne kontrollere og systemer. Plattformen er også utstyrt med en berøringsskjerm som gjør det enkelt for andre å justere og endre parametere for sine kontrollere. Alle skriptene er programmert i Python og er modulære, slik at hvert skript kan erstattes. For å oppfylle kravene til beregningskraft, brukes en Khadas VIM3 som systemet for all programvare. Dette gjør at MPC enkelt kan styre en ball på toppen med minimale feil.

Contents

List of Figures

List of Tables

Acronyms

MPC Model Predictive Control

FOV Field of View, the angle at which something is observable.

GUI Graphical User Interface, makes it possible to interact with a computer.

DOF Degrees of Freedom, number of configurations for an object.

LP Linear Programming, formulation for solving MPC problem.

QP Quadratic Programming, formulation for solving MPC problem.

DCP Disciplinary Convex Programming.

DPP Disciplinary Parameterized Programming.

HSV Hue, Saturation, Value, alternate representation of a colour space.

CPU Central Processing Unit.

UART Universal Asynchronous Receiver-Transmitter.

OS Operating System.

SSH Secure Shell.

PID Proportional–Integral–Derivative controller.

GPIO General-purpose input/output.

Hz Hertz.

Nomenclature

$x(i)$ System state in the MPC.

$u(i)$ System input in the MPC.

Q Weighting matrix for state variables in MPC.

R Weighting matrix for input variables in MPC.

J Cost function for MPC.

Δt Sampling time.

H Horizon length in MPC.

n Number of state variables in MPC.

m Number of input variables in MPC.

x_{ref} Reference states in the MPC cost function.

u_u **and** l_u Input bounds in the MPC.

$x_{initial}$ Initial condition for the states in the MPC.

ϕ Platform angle along the X-axis.

θ Platform angle along the Y-axis.

α Platform angle in the simulation-sketch.

β Motor angle in the simulation-sketch.

p The length from the centre to the platform leg.

f The length of the motor leg.

Chapter 1

Introduction

To gain a deeper understanding of control systems, hands-on experience with practical and real-world systems is necessary. In this thesis, we seek to fill this gap by designing and building a motion platform suitable for educational purposes. The motion platform is an excellent tool for students to simulate real-world scenarios in a controlled environment for testing and experimenting. In recent years a powerful control strategy called Model Predictive Control has gained popularity due to its application for high-performance control, and its ability to handle complex nonlinear systems with constraints. The MPC has also been seen as a more viable option due to recent improvements in algorithm development. The MPC calculates the appropriate control action by using measurements, and a dynamical model of the system to predict future states. Due to its computational requirements, and complex nature, there is a lack of good practical educational tools to experiment with using this control strategy. This motion platform is designed to educate students and provide a hands-on learning experience with MPC, and other control strategies, to allow them to gain a deeper understanding of control systems.

1.1 Background

A motion platform is a mechanical system, often referred to as a motion simulator or motion system, that simulates the motion and forces people and items might experience in a given environment or circumstance.

Motion platforms are widely used in multiple fields, including entertainment, military, aviation, and technical research. For example, in motion theatres, or vehicle simulators. They can mimic the motion of different moving objects, including cars, ships, and planes. In addition to being used for entertainment and training applications, motion platforms can also be used in research and development to study the consequences of motion on human physiology and performance.[11]

The dynamic behaviour of a motion platform can be described by a mathematical model of the system that describes an object and platform's position, velocity, and acceleration in relation to the forces and torques acting on it. These models can be written in the form of differential equations or state-space models, which is a mathematical representation of the system in terms of its states, inputs, and outputs. Because of their wide range of use and complexity, motion platforms are an ideal teaching tool for students studying control theory.

1.2 Problem Formulation

The main problem formulation for our project is to design and build a 3 degrees of freedom motion platform that is robust and user-friendly, to be used as a testing ground for students to learn and experiment with various controllers including Model Predictive Control.

The problems addressed throughout this project are derived from two main objectives. The first objective is to design and assemble a motion platform that others can easily modify to suit their own objectives. The second objective is to explore the possibilities of designing an MPC that supplements the motion platform.

These two main objectives form the foundation for further tasks and problems addressed throughout the project. To ensure these main objectives are accomplished to an acceptable degree the following problems are to be addressed.

Goals to achieve

1. Design and assemble a stable and robust 3DOF motion platform.

2. Design and assemble a compact enclosure, with easy access, to house all necessary components.

3. The motion platform should fit in a typical shelf, and be stackable. Given that shelves have various sizes the width of the motion platform should be less than 60 cm.

4. Model and program a model predictive controller to balance a ball for the motion platform.

5. Create a graphical user interface that allows users to experiment with different approaches and tuning.

6. Ensure the software is easy to modify with the aim to allow users to make their own controllers.

1.3 Related Work

This report is mainly based on these publications:

- Design and Application of a Motion Platform in Three Degrees of Freedom[9]. This publication presents a 3DOF motion platform which is used to balance a ball by using servo motors. Presents thorough documentation of the mathematical models and angle transformations between the X and Y-axis to the 3DOF platform.

- Fast Real-Time Model Predictive Control for a Ball-on-Plate Process[17]. This publication uses a 2DOF motion platform that incorporates MPC to control a ball. A thorough demonstration of how to incorporate MPC in a ball-balancing platform. Discusses multiple approaches and methods to achieve good results.

- 3DOF Ball on Plate Using Closed Loop Stepper Motors[8]. This publication includes a step-by-step guide on how to construct a 3DOF motion platform that incorporates stepper motor control and robust design. Presents design approaches and demonstration of how to achieve a stable and robust construction.

1.4 Scope

Because of the limited time frame and the number of possible approaches to designing motion platforms, certain limitations are imposed on the functionality.

- The platform is expected to have the capability of three degrees of freedom, but the controller will only incorporate roll and pitch functionality.
- The control algorithm should be simplified by using certain approximations and linearizations due to the complex nature of the mathematical models underlying a motion platform.

1.5 Structure of the Report

The rest of the report is structured as follows.

Chapter 2 - Preliminaries: Chapter two gives an introduction to the theoretical background and preliminaries used to model, construct and develop the system, controller, and software for the 3DOF motion platform.

Chapter 3 - Method: Contains a description of the development of the platform, and which solutions were considered and attempted throughout the development to arrive at the desired result.

Chapter 4 - Result: Contains a presentation of the finished product and performance tests.

Chapter 5 - Discussion: A summary and evaluation of the performance and results, the utilization, and which changes can be made to further improve the product.

Chapter 6 - Conclusions: This chapter present an overall conclusion of the 3DOF motion platform

Chapter 2

Preliminaries

The project's underlying foundations will be covered in this chapter. It will outline the software applications used to create the project as well as the theoretical approach taken to solve the problem.

2.1 Software

There were several programs and special software libraries used throughout the project. The section below includes a list of these.

2.1.1 PyCharm

PyCharm is an integrated development environment (IDE) used for coding Python. There are multiple features that improve the efficiency of coding and troubleshooting. It provides a code editor with syntax highlighting, code completion, and error highlighting. It also offers custom tools for debugging, testing, profiling, and integration of version control. The software is available in two editions: the Community edition and the Professional edition. The software is user-friendly and provides a helpful user guide.

2.1.2 Arduino IDE

Arduino IDE (Integrated Development Environment) is a software program used to develop and upload code to Arduino micro-controller boards. The program allows users to create interactive electronic projects using the open-source electronics platform Arduino. The Arduino IDE offers libraries and functions made specifically for Arduino boards, and it supports the C and C++ programming languages. As a result, beginners don't need to have a strong background in programming to get started creating electronic projects.

2.1.3 Autodesk Fusion 360

Autodesk Fusion 360 is a cloud-based 3D computer-aided design software application that is used for product design, engineering, and manufacturing. Designers and engineers can use Fusion 360 to build 3D models and parts assemblies, as well as 2D drawings based on those models. It is a complete software solution for product development because it also offers tools for collaboration, generative design, and simulation.

2.1.4 Pruca Slicer

Prusa Slicer is a slicer software for 3D printing. It is an open-source, feature-rich, and frequently updated tool that contains everything you need to export print files to a Prusa 3D printer. To guarantee the greatest outcome possible, the application offers parameter modifications for infill, print quality, and customizable supports.

2.1.5 Geogebra

GeoGebra is a free and open-source mathematics software program that allows users to create and manipulate dynamic geometrical constructions, as well as perform algebraic computations, statistics, and calculus. Users can drag and manipulate objects in real-time, and observe the effect of changes on other objects in the construction.

2.1.6 EPLAN Electric

EPLAN Electric P8 is an electrical schematic software where the user can plan and design electrical drawings. It supports the use of I/O lists and can directly wire components by itself. Manual drawing is also supported by the software, together with standardized and template-based methods. The software allows the user to have large projects that can utilize both 2D and 3D formats. The program is very technical, but courses and guides are provided for the user.

2.1.7 Flexi Designer Flexi

Flexi Designer is a program that is used for laser-cutting purposes. It allows the user to import 2D sketches and modify them if needed. When the sketch is finished, it gets exported to the desired laser-cutting equipment. The laser cutter will follow the 2D lines which were displayed in the software. The software is easy to use and requires little training.

2.1.8 Programming libraries

CVXPY

CVXPY is an open-source Python package used for Python-embedded modelling of convex optimization problems. CVXPY allows users to formulate convex optimization problems in a mathematical and natural way using Python syntax, rather than using the strict forms other solvers require. It supports a wide range of convex optimization problems, including linear programming, quadratic programming, semidefinite programming, and cone programming. CVXPY is built by a community of researchers, engineers, and students across the world.[4][3][5]

CVXPY is licensed under the Apache License, Version 2.0

CVXPYgen

CVXPYgen is an add-on library for the CVXPY python package. It is built on top of the CVXPY model and allows users to generate custom solvers in the C language by defining the problem parameters in Python. It is designed to create easy and efficient solvers for convex optimization problems. The generated code uses low-level optimization techniques to achieve high performance. CVXPYgen is particularly useful in situations where standard convex optimization solvers do not provide the desired performance, or when custom optimization algorithms are needed for specialized applications.[16]

CVXPYgen is licensed under the Apache License, Version 2.0

OpenCV

OpenCV is an open-source computer vision and machine learning software library that is designed to help developers create real-time applications for image and video processing. It provides a wide range of algorithms for image processing and computer vision, including object detection and tracking, image segmentation, feature detection, and optical flow analysis.

OpenCV was originally developed by Intel Corporation in 1999 and is now maintained by the OpenCV Foundation, a non-profit organization with the goal of promoting and advancing computer vision research. It is available for interfacing in C++, Python, Java, and MATLAB, and supports Windows, Linux, Android, and Mac OS.

OpenCV 4.5.0 and higher versions are licensed under the Apache 2 License. While OpenCV 4.4.0 and lower versions are licensed under the 3-clause BSD license.

PyQt5

PyQT is a Python library that is used to create graphical user interface applications with the use of the QT tool. The library is free and open source and has been in development since 1999. The newest version is PyQT5 which was released in 2016. The library provides a wide range of classes and modules which are used to create the applications. Its use of widgets makes it flexible and easy to use and this is why the library is very popular. PyQT5 also supports integration with other Python libraries such as NumPy and Matplotlib.

PyQT5 is developed by Riverbank Computing and is licensed under GNU GPL v3.

Other minor packages

Table 2.1 Summarizes all preliminary packages used in the thesis. The Table lists version numbers and a short description of each package.

Package	**Version**	**Description**
platformdirs	3.2.0	Package for determining platform-specific directories.
numpy	1.23.24	Mathematical package, useful for array and matrix calculations.
time	N/A	Package for determining time.
datetime	N/A	package for determining both date and time.
pickle	N/A	Package for converting byte-stream.
sys	N/A	Package with system-specific parameters and functions.
threading	N/A	Multi-thread package, run multiple parts of a script concurrently.
multiprocessing	N/A	Similar to threading runs scripts concurrently as subprocesses instead of threads.
subprocess	N/A	Package for spawning new process.
pyserial	3.5	Package for serial communication.
struct	N/A	Package for interpreting bytes.
atexit	N/A	Cleanup package for terminating functions.

Table 2.1: Preliminary package table.

2.1.9 Version Table

Table 2.2 lists the version numbers of all software.

Software	Version
Arduino IDE	2.1.0
Autodesk Fusion 360	2.0.15995
Prusa Slicer	2.5.0
Geogebra Classic	5.0.775.0
EPLAN Electric	2023
Flexi Designer	12
CVXPY	1.3.1
CVXPYgen	0.2.2
OpenCV	4.7.0
PyQt5	5.15.9

Table 2.2: Software version table.

2.2 Degrees Of Freedom

Degrees of freedom is a term that describes the number of independent parameters used to describe the configurations of a robot. The position and orientation of the body can be described by three components of translation and three components of rotation. When a body can be described by all six of these components, it is said to have six degrees of freedom shown in Figure 2.1[1]. These components contain the following:

- Rotation
  - Roll - Tilting side to side along the X-axis.
  - Pitch - Tilting forward and backwards on the Y-axis.
  - Yaw - Turning left and right on the Z-axis
- Translation
  - Surge - Moving forward and backwards on the X-axis.
  - Sway - Moving left and Right on the Y-axis.
  - Heave - Moving up and down on the Z-axis

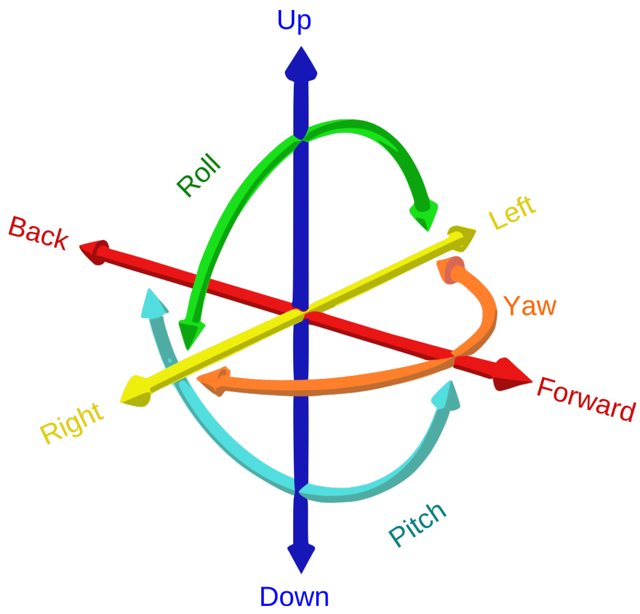


Figure 2.1: Figure illustrating all six degrees of freedom[10].

The motion platform considered in this thesis has three degrees of freedom: roll, pitch, and heave. We therefore refer to this platform as a 3DOF motion platform.

2.3 Model Predictive Control

Model Predictive Control (MPC) is a control strategy that solves a constrained optimization problem. The MPC solves the problem at each control interval to find the optimal control actions that minimize a cost function subject to system dynamics and input constraints. The MPC predicts and optimizes future process behaviour subject to the cost function and constraints over a predicted time horizon. MPC is a widely used technique in control engineering because it can handle nonlinear systems, constraints, and disturbances.[6]

The optimization problem for the MPC is typically cast into one of two forms. Either a linear Programming formulation or a quadratic programming formulation. In an LP formulation, both the cost function and constraints are linear. In a QP formulation, the cost function is quadratic while the constraints are linear. Additionally, a QP requires that the problem is convex such that a unique optimal solution can be found quickly.[6]

An example of a QP optimization problem that typically arises in MPC applications is given by:(2.1)

$$
\begin{aligned}
\text{Minimize:} \quad & J = \sum_{i=0}^{N-1} \left[(\mathbf{x}(i) - \mathbf{x}_{ref})^T \mathbf{Q} (\mathbf{x}(i) - \mathbf{x}_{ref}) + \mathbf{u}(i)^T \mathbf{R} \mathbf{u}(i) \right] \\
\text{Subject to:} \quad & \mathbf{x}(i+1) = \mathbf{A}\mathbf{x}(i) + \mathbf{B}\mathbf{u}(i), \quad i = 0 \dots N-1 \\
& \mathbf{x}(0) = \mathbf{x}_{initial} \\
& \mathbf{l}_u \le \mathbf{u}(i) \le \mathbf{u}_u \\
& \mathbf{x}(i) \in \mathscr{X}, \quad i = 1 \dots N
\end{aligned}
\tag{2.1}
$$

$\mathbf{x}(i)$	is the system state at time step i.
$\mathbf{u}(i)$	is the control input at time step i.
N	is the prediction horizon.
$\mathbf{x}_{ref}$	is the reference state.
$\mathbf{Q}$ and $\mathbf{R}$	are positive definite weighting matrices.
$\mathbf{A}$ and $\mathbf{B}$	are the state and input matrices of the system model.
$\mathbf{l}_u$ and $\mathbf{u}_u$	are the lower and upper bounds on the control inputs.
$\mathscr{X}$	is the feasible region of the state variables.

2.3.1 Tuning MPC

Several parameters are required to be specified to design an MPC controller. All of these parameters can be used to tune and change the response of the controller. Defining an MPC controller requires a sampling period Δt and a model horizon H. The sampling period determines the length of time between each update of the system states and inputs of the predicted state and control variables. The horizon length determines how far into the future the controller updates these variables. For example, a sampling time of 0.1 seconds and a horizon length of 10 updates the state and control variables every 0.1 seconds 1 second into the future. Typically a smaller value for H results in more aggressive control.

The other parameters used to tune an MPC controller are the weighting matrices Q and R. These matrices determine the amount of weight used in the state and input variables when calculating the optimal input variable. The Q matrix has a size of $n \times n$ where n is the number of states, and the R matrix has a size of $m \times m$ where m is the number of input variables. The Q matrix has weights across the diagonal where each number corresponds to the amount of weight placed on each state variable. The R matrix follows the same principle with weights placed across its diagonal, where each number correspond to the amount of weight placed on each input variable. By tuning these numbers, one can easily decide which variable the controller will react more aggressively to. Typically a larger Q equals more aggressive control, and a larger R equals more conservative control.[12]

2.4 Disciplined Convex Programming

Disciplined Convex Programming is a framework for writing optimization problems with a set of rules and guidelines. This way of formulation allows for efficient and reliable optimization using convex programming techniques. Convex optimization is a powerful mathematical tool for solving a wide range of problems in engineering, economics, and other fields. A convex optimization problem is one where the objective function and the constraints are all convex functions.

The set of rules the DCP framework provides allows the optimization problem to be automatically checked for convexity. The rules specify how mathematical expressions can be combined, what operations are allowed, and what functions are convex. By following these rules, one can be sure that the resulting optimization problem is convex, which means that it has a unique global minimum.[4][3][5]

2.4.1 Disciplined Parameterized Programming

Disciplined Parameterized Programming is a ruleset for producing parameterized DCP-compliant problems. In DPP parameters are used to define the behaviour of a program, and the program is designed to work correctly for any valid input values of the parameters. The first time a DPP parameterized problem is compiled, the program caches the mapping from parameters to problem data. This allows further iterations of the program to run substantially faster, even when changing the parameters of the problem. [4][3][5]

Both the DCP and DPP rulesets and concepts are included in the CVXPY and CVXPYgen programming libraries.

2.5 HSV Colour Space

HSV refers to an alternate representation of colour space which represents the colour based on how the human eye observes colour. This colour space represents the colour in three layers respectively. These layers are hue (H), saturation (S), and value (V). Hue is the attribute that represents the wavelength, or pureness of the colour such as red, green, or blue. Saturation is the attribute that describes the intensity, or purity of the colour which is diluted by white light. Value is the attribute that represents the brightness of the intensity for the colour. These layers of attributes can be represented in a three-dimensional model as seen in Figure 2.2.[14]

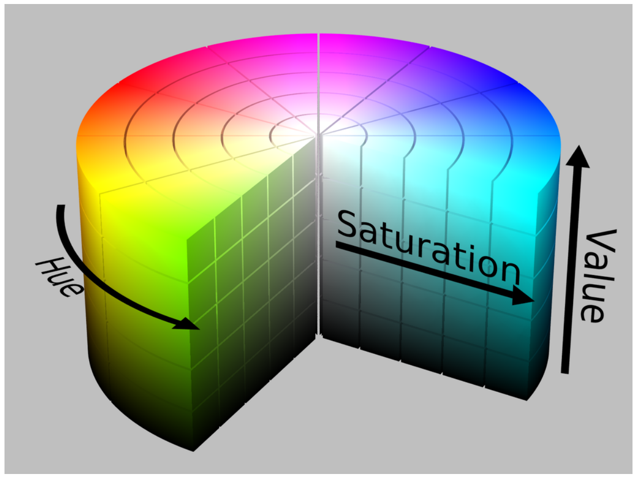


Figure 2.2: three-dimensional model representation of HSV colour space.[7].

The HSV colour space is often used in image editing software for adjusting the colour and brightness of images. It is also used in computer vision applications such as object recognition and tracking, where it can be useful for isolating objects based on their colour.

HSV colour space is often, but not always, used in colour detection and tracking applications in the computer vision industry, where it can be used to isolate objects based on their colour. It is also widely used in image editing software to adjust the colour and brightness of images. Because HSV colour space has adjustable brightness and intensity attributes, it is quite robust under different light conditions.

2.6 Shared Memory

Shared memory is a technique used in programming that allows multiple processes or threads to access the same memory. This means that multiple processes can easily share variables without the need for inter-process communication techniques such as pipes or messages.

In a shared memory system, a process can allocate memory within a shared memory segment. Other processes can then access this shared memory segment at the same time, and access the memory another process has allocated. This enables processes to collaborate without synchronization or copying mechanisms. The shared memory technique is a simple way to communicate variables and information streams between processes and eliminates the need for more complicated and time-consuming techniques. The downside to using shared memory is that when a large number of processes try to access the same memory segment, issues can occur.[13]

Chapter 3

Method

This chapter describes how the platform was designed, wired, and how the code was implemented. It will also provide sufficient technical documentation to simplify how the project was solved.

3.1 Design and assembly

This section describes the design and assembly of the motion platform. There are multiple factors to consider when making the platform to ensure it is robust and stable, such that mechanical error is minimized. The most important part is a robust foundation. Figure 3.1 shows the initial design sketch for the platform.

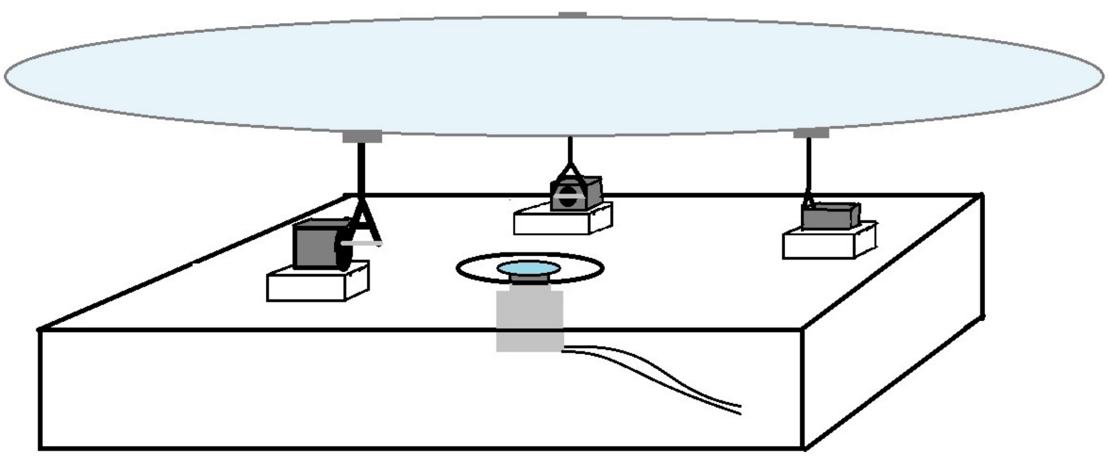


Figure 3.1: First sketch of the platform.

3.1.1 Foundation and enclosure

The foundation consists of a rectangular bottom plate which is 60cm x 41 cm. 3D-printed columns are placed in each corner to act as a robust framework to which the side plates and top plate can be attached. On the furthest end of the bottom plate, triangle supports are attached to hold the plate for the GUI screen. The side plates are laser cut with a square-shaped mesh, such that fans can provide sufficient cooling. All plates are laser cut from 6 mm MDF. This is because MDF is the cheapest and most available option. The foundation is shown in Figure 3.2.

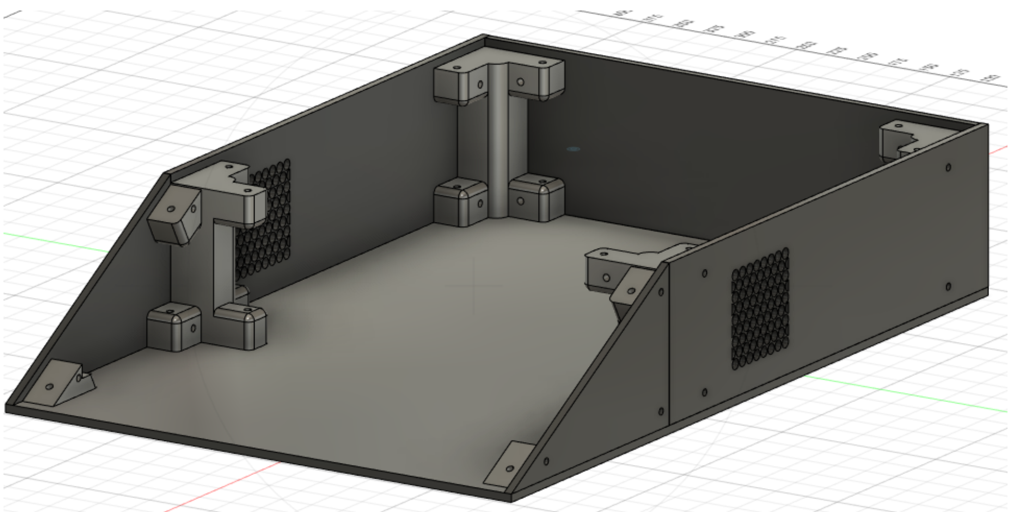


Figure 3.2: Fusion design of the foundation.

To make the total enclosure a top plate and screen plate are attached on top of the foundation. The screen plate is a simple frame in which a rectangular hole exists to fit the screen perfectly. This allows all cables coming from the screen to be concealed on the inside of the enclosure. The top plate is attached on top of the column framework, and includes holes for each stepper motor mount to be attached. A small round hole is placed in the centre of the plate. this hole is designed to let the camera observe the platform above, while at the same time being concealed. The total dimension for the enclosure is 60 cm × 41 cm × 12.2 cm. The enclosure is shown in Figure 3.3.

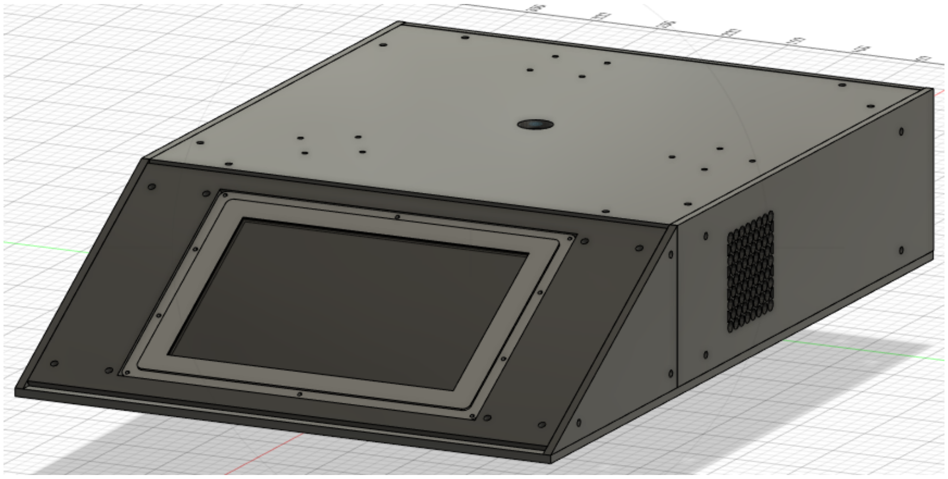


Figure 3.3: Fusion design of the enclosure.

3.1.2 Platform

The platform and motors are placed on top of the enclosure. This is to make sure all wiring and cables can be concealed and passed easily inside the enclosure. The stepper motors are placed on custom-designed mounts which are 3D-printed in PLA. The mounts are hollow on the inside to ensure all cables can be concealed. The platform is laser cut from acrylic to allow the camera to observe the ball. The platform is designed with protrusions to attach the joints. This is to make sure the joints are attached without obstructing the view of the camera. The joints are attached to the protrusion using 3D-printed intermediary bricks. This is done to minimize friction and allow easy mounting for the ball joints. The platform is shown in Figure 3.4.

The leg joints for the platform consist of one ball joint at the top and two ball joints at the bottom. To ensure these ball joints are stable, and to minimize flexing, legs in the form of an *A* are attached. From the bottom ball joints to the motor a 3D-printed leg is attached. This motor leg clamps down on the motor axle to make sure the leg does not slide. The leg is also very thick to reduce the amount of flexing and trembling. A close-up of the leg joint is shown in Figure 3.5.

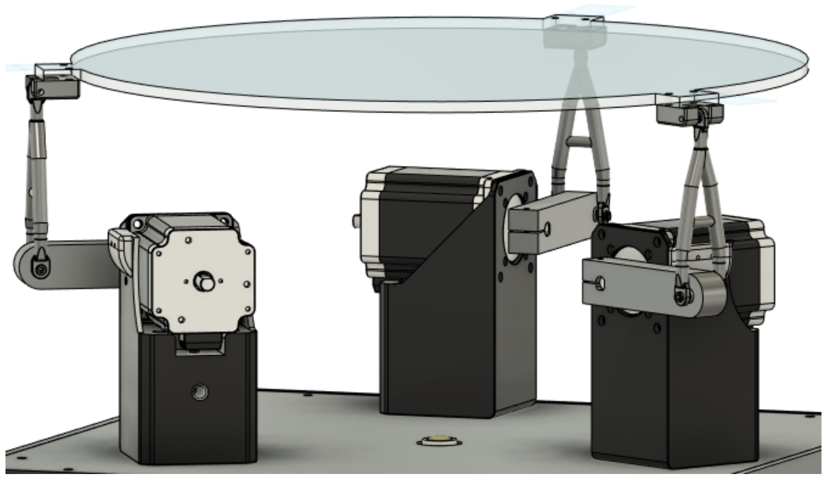


Figure 3.4: Fusion design of the platform.

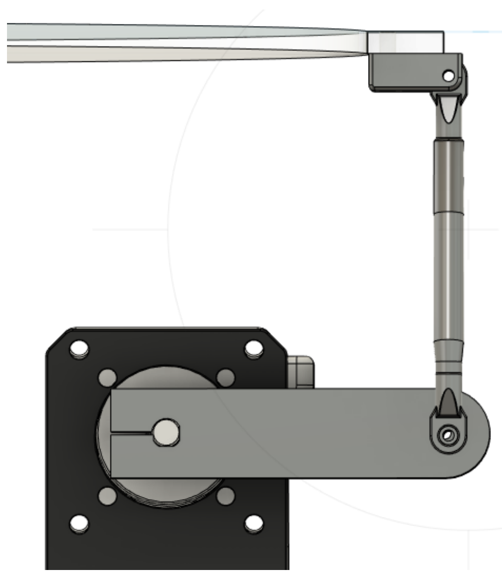


Figure 3.5: Fusion design of the joint between the motors and platform.

3.1.3 Practicality

In order for the design to achieve goal 2, every component is placed and integrated together inside one box. This makes the device easy to transport and store. When shut off the dimensions of the motion platform are 60 cm × 41 cm × 26 cm to 60 cm × 41 cm × 33.1 cm when operating. This makes it compact enough to store in shelves or other storage locations. The box also serves as a stable and robust foundation for the motors and platform to operate on. When the device is shut off, the platform descends to a resting position on top of the motors. This decreases its height and allows the device to be stacked with other inventory.

This device is designed for easy recreation to ensure that it can be used to its fullest potential in education, and for other students to make their own on campus. The parts are either laser cut in 4mm and 6mm MDF and acrylic, or 3D printed in PLA. All parts are easy to obtain, widely available, and could be provided as a kit, all parts are listed in Table 3.1 and 3.2. All integrated components are attached with bolts, and the holes are laser cut or 3D printed. All this adds up to a simple build which is easy to recreate and modify.

3.1.4 Integration

A great variety of components are integrated together to run and operate the platform. All components which are listed in the parts Table 3.2 had to be located inside the box and be easy to access. Since the stepper motor drives run on 48V, a 230V AC to 48V DC converter had to be installed. Having 3 stepper motors running independently requires 3 motor drives. The Khadas card was not able to handle interrupts from the encoders, which lead to the need to add an Arduino. The camera also needs a mount to be in the correct position to ensure visibility of the whole platform. The touchscreen for the GUI is placed in a convenient position for easy use. The original power adapters are used for the Khadas and touchscreen to avoid noise and other compatibility issues. Therefore two 230V sockets were installed. To prevent any form of overheating issue, mainly from the Khadas VIM, 2 fans are placed strategically to process the heat. One for inlet, and one for outlet. The fans operate on a 12V DC supply, which demands a 230V AC to 12V DC supply. In addition to this, a user-friendly solution is established for the wires coming from each motor. This is done by mounting 12-pin plugs for each motor. All electrical components are presented in the component overview 3.7 and the electrical schematic drawing 3.6. A list of all bolts which are used to construct the platform is listed in Table 3.1. A list of all parts which are used to construct the platform is listed in Table 3.2.

Size/Length	12mm	16mm	20mm	25mm	30mm	35mm	40mm
M2		4	6				
M3			15		3		3
M4	12		6	12			
M5							
M6		2	4				44

Table 3.1: Amount of bolts used.

Qty	Description	Manufacturer	Part Number	Mounting Method
1	Wide Angle USB Camera	Arducam	B0202	Bolts
1	Wide Angle Lens	Arducam	M25170H12	Threads
1	Arduino Mega	Arduino	Mega 2560 Rev3	Tape
3	Modular Incremental Encoder	CUI DEVICES	AMT102-V	Bolts
3	Shielded encoder cable	CUI DEVICES	CUI-3131-6FT	Plug
1	KHADAS VIM3	KHADAS	VIM3 Amlogic 311D SBC	Tape
1	230V-12V DIN Rail Power Supply	MEAN WELL	HDR-15-12	DIN-Rail
1	230V-48V/5V Power Supply	MEAN WELL	ADS-15548	Bolts
3	12 PIN Plug Female	PTR HARTMANN	AKZS 12	DIN-Rail
3	12 PIN Plug Male	PTR HARTMANN	AKZ950/12-5.08-GRÜN	On Wire
3	2-Phase Digital Stepper Drive	Stepperonline	DM556T	Bolts
3	Nema 23 Dual Shaft Stepper Motor	Stepperonline	23HS30-2804D	Bolts
2	12V 80x80x25 Fan	Sunon	EF80251S2-1000U-A99	Bolts
2	Wago Electrical Outlet	WAGO	709-583	DIN-Rail
1	10.1 inch Touch Display	Waveshare	WS3-011750-P	Bolts
3	Limit Switch with Hinge Roller Lever	RND Components	RND 210-00725	Bolts
1	Bi Directional Logic Level Converter	Sparkfun	BOB-12009	Breadboard
1	Breadboard	RND Components	RND 255-00020	Tape
1	40cm DIN RAIL	Weidmuller	514510000	Bolts

Table 3.2: Part list.

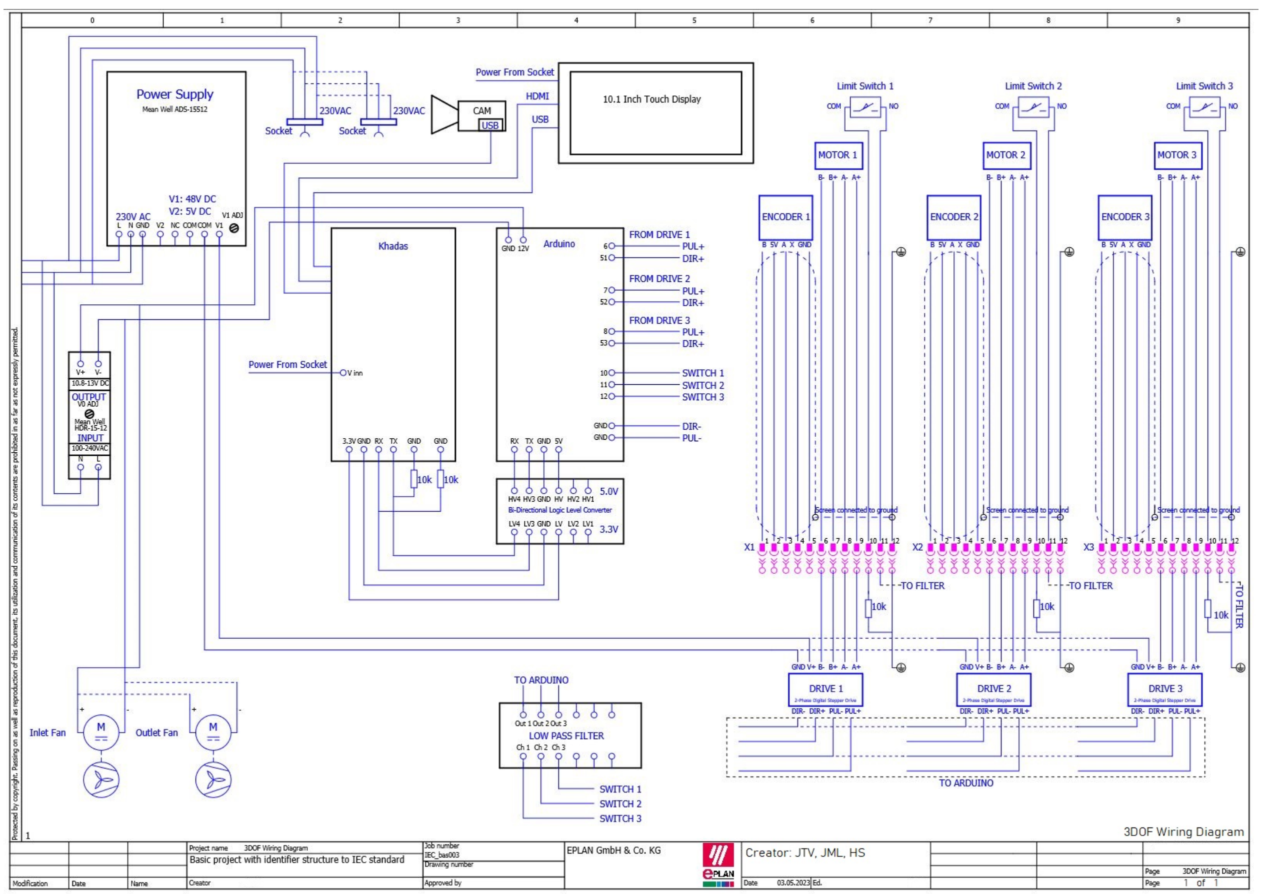


Figure 3.6: Electrical drawing of the System.

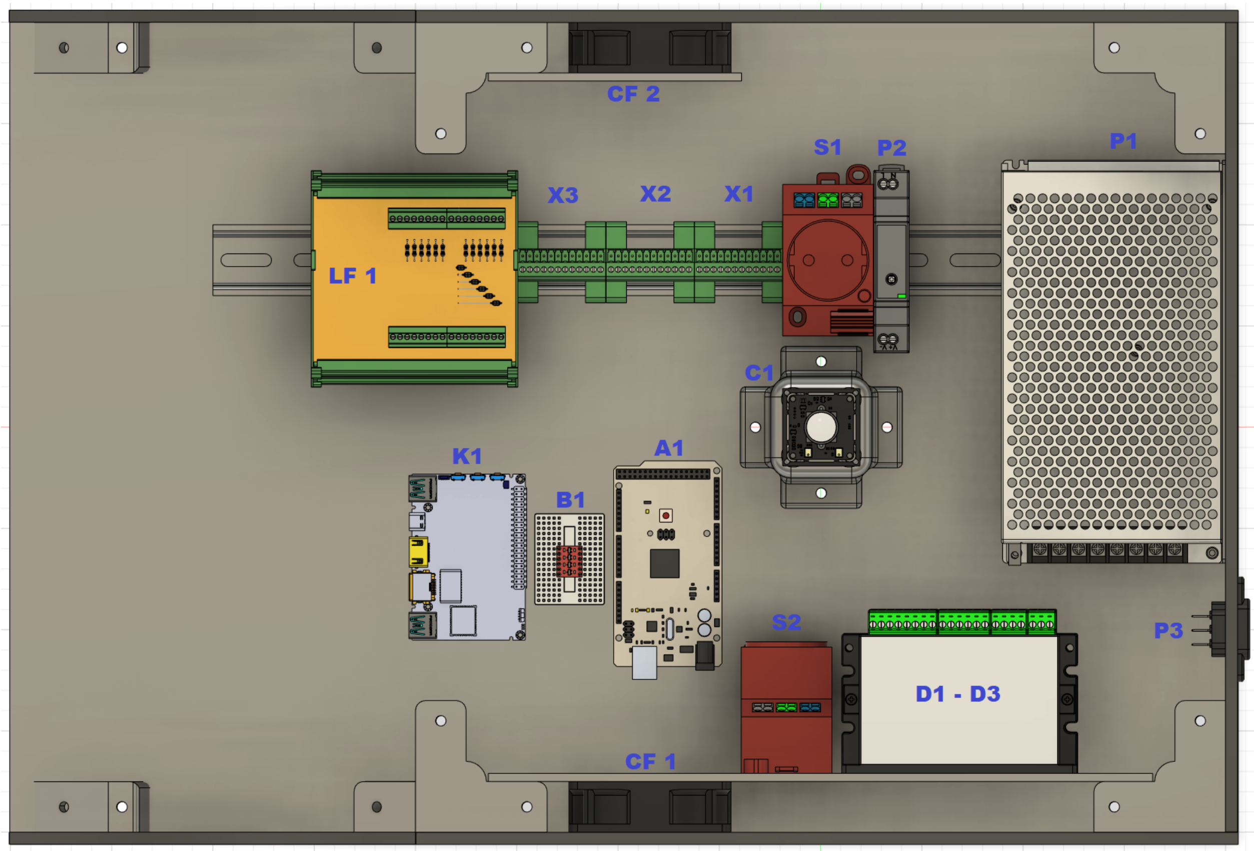


Figure 3.7: Component Layout Overview.

ID	Meaning	Description
LF 1	Low Pass Filter	A custom built low pass filter for handling noise
X1	Plug	12 Pin plug for connecting Motor 1, Limit switch 1 and Encoder 1
X2	Plug	12 Pin plug for connecting Motor 2, Limit switch 2 and Encoder 2
X3	Plug	12 Pin plug for connecting Motor 3, Limit switch 3 and Encoder 3
S1	Socket Outlet	Socket Outlet for powering the GUI Touch Display
S2	Socket Outlet	Socket Outlet for powering the Khadas (K1)
P1	Power Supply	Power Supply converting 230VAC to 48VDC to power the motor drives (D1-D3)
P2	Power Supply	Power Supply converting 230VAC to 12VDC to power the cooling fans (CF1 & CF2)
P3	Power Plug	IEC Female power connector for 230VAC
K1	Khadas	Khadas single board computer to run the code
B1	Converter	Bi-Directional Logic Level Converter for converting 5VDC to 3.3VDC between the Khadas and Arduino (K1 & A1)
CF1	Cooling Fan	Inlet Cooling Fan to manage high temperatures generated by the Khadas card (K1)
CF2	Cooling Fan	Outlet Cooling Fan to manage high temperatures generated by the Khadas card (K1)
A1	Arduino	Arduino Mega which handles the stepper motor signals to the motor drives (D1-D3) and passes it to the Khadas (K1)
C1	Camera	Wide lens camera to track ball position and send data to the Khadas (K1)
D1	Drive	Stepper Motor Drive 1 which controls motor 1 by communication with the Arduino (A1). (Top position in Figure)
D2	Drive	Stepper Motor Drive 2 which controls motor 2 by communication with the Arduino (A1). (Middle position in Figure)
D3	Drive	Stepper Motor Drive 3 which controls motor 3 by communication with the Arduino (A1). (Bottom position in Figure)

Table 3.3: Table explaining location of each part.

3.2 Stepper Motors

This section describes which stepper motors were chosen to control the platform, and how they were implemented.

3.2.1 Motor Specifications

The stepper motors that are used, were of the type Nema 23HS30-2804D 3.2, which can be observed in the provided data sheet 3.8. It is a bipolar 2-phase stepper motor which runs on 48V from the motor drives, which again are powered from the ADS-15548 Power supply 3.6. Each step angle equals 1.8 degrees of rotation on the shaft. At peak torque, it can deliver 1.8 N m. The torque needed was calculated by a simple set of formulas:

The volume of the top plate is given by:

$$\pi \cdot 20cm^2 \cdot 0.6cm = 753cm^3 \tag{3.1}$$

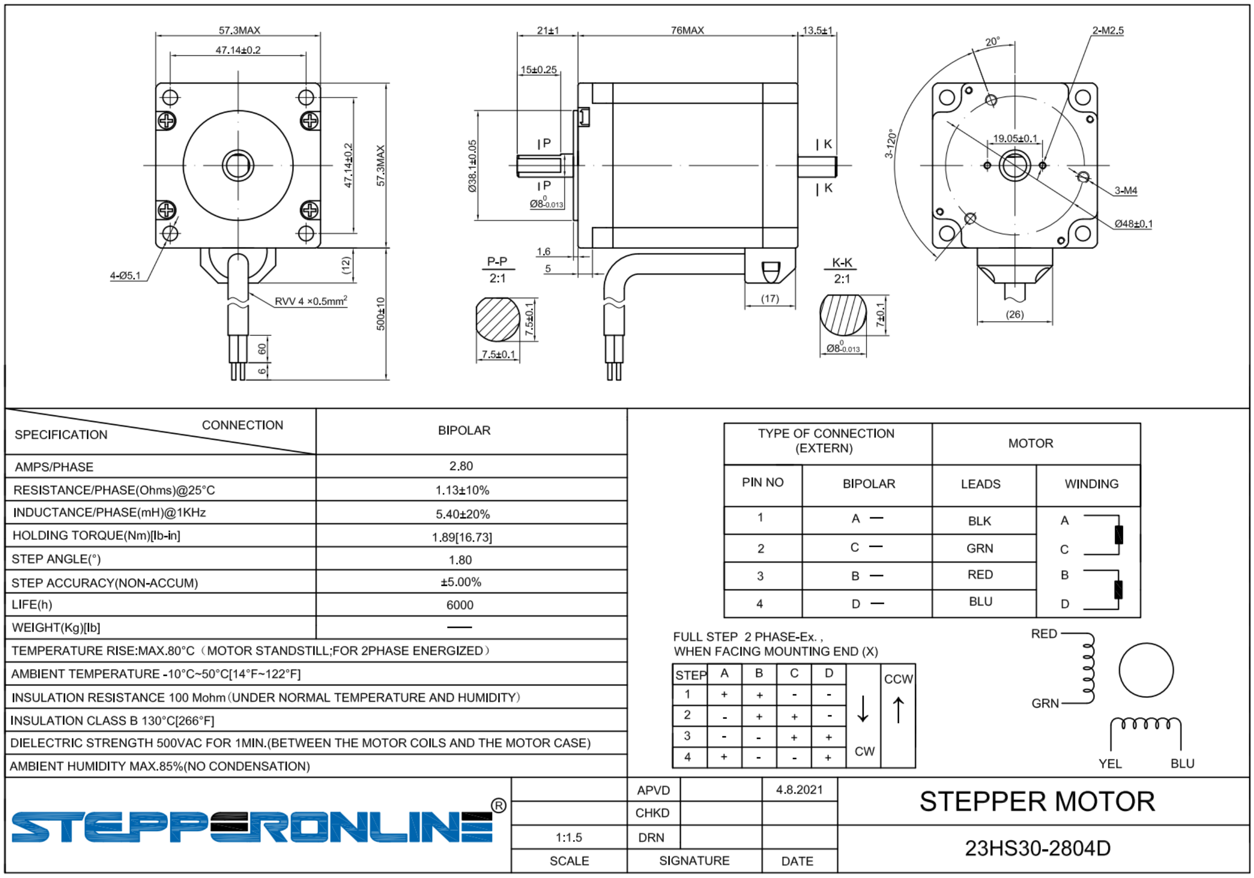
Given that the density of acrylic sheet is $1.18\,\mathrm{g\,cm^{-3}}$ the weight and force of the plate becomes:

$$\begin{aligned} 753cm^3 \cdot \frac{1.18g}{cm^3} = 0.8897kg \\ 0.8897kg \cdot 9.81m/s^2 = 8.7280N \end{aligned} \tag{3.2}$$

Given that 3 motors are used, the minimum holding torque of each motor with a motor leg length of 10 cm becomes:

$$\begin{aligned} \frac{8.7280N}{3} = 2.9093N \\ 2.9093N \cdot 0.1m = 0.291Nm \end{aligned} \tag{3.3}$$

A minimum requirement of 0.6 N m has been set to account for fast movements and design changes. This made the Nema 23HS30-2804D a good choice because of price and specifications as seen in Figure 3.8



SPECIFICATION \ CONNECTION	BIPOLAR
AMPS/PHASE	2.80
RESISTANCE/PHASE(Ohms)@25°C	1.13±10%
INDUCTANCE/PHASE(mH)@1KHz	5.40±20%
HOLDING TORQUE(Nm)[lb-in]	1.89[16.73]
STEP ANGLE(°)	1.80
STEP ACCURACY(NON-ACCUM)	±5.00%
LIFE(h)	6000
WEIGHT(Kg)[lb]	—
TEMPERATURE RISE:MAX.80°C (MOTOR STANDSTILL;FOR 2PHASE ENERGIZED)	
AMBIENT TEMPERATURE -10°C~50°C[14°F~122°F]	
INSULATION RESISTANCE 100 Mohm (UNDER NORMAL TEMPERATURE AND HUMIDITY)	
INSULATION CLASS B 130°C[266°F]	
DIELECTRIC STRENGTH 500VAC FOR 1MIN.(BETWEEN THE MOTOR COILS AND THE MOTOR CASE)	
AMBIENT HUMIDITY MAX.85%(NO CONDENSATION)	

TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A —	BLK	A
2	C —	GRN	C
3	B —	RED	B
4	D —	BLU	D

STEP	A	B	C	D
1	+	+	-	-
2	-	+	+	-
3	-	-	+	+
4	+	-	-	+

STEPPERONLINE®		APVD		4.8.2021
		CHKD		
	1:1.5	DRN		
	SCALE	SIGNATURE		DATE

STEPPER MOTOR
23HS30-2804D

Figure 3.8: Data Sheet for stepper motors [2].

3.2.2 Motor Drives

The stepper motors were operated by the DM556T Digital Stepper Drives shown in Table 3.2. These ensured great flexibility for speed and smoothness by having dip-switch options to micro-step. This feature ranged from 400 to 25600 steps per rotation of the shaft shown in Figure 3.9. The motor drives are connected to the Arduino for communication.

Microstep	Steps/rev.(for 1.8°motor)	Sw5	SW6	SW7	SW8
2	400	OFF	ON	ON	ON
4	800	ON	OFF	ON	ON
8	1600	OFF	OFF	ON	ON
16	3200	ON	ON	OFF	ON
32	6400	OFF	ON	OFF	ON
64	12800	ON	OFF	OFF	ON
128	25600	OFF	OFF	OFF	ON
5	1000	ON	ON	ON	OFF
10	2000	OFF	OFF	ON	OFF
20	4000	ON	OFF	ON	OFF
25	5000	OFF	OFF	ON	OFF
40	8000	ON	ON	OFF	OFF
50	10000	OFF	ON	OFF	OFF
100	20000	ON	OFF	OFF	OFF
125	25000	OFF	OFF	OFF	OFF

Figure 3.9: Dip switch Setting for motor drives [2].

3.2.3 Control

The motors, and motor drives, are controlled by an Arduino. The Arduino sends a pulse to define how many steps to complete and the direction of the rotation. The speed is determined by the *SetSpeedInHz* variable and the acceleration is determined by the *SetAcceleration* variable shown in Figure 3.12. The Arduino receives its input values from the Khadas running the main scripts. This signal goes through a 3.3V to 5.0V Bi-Directional Logic Level Converter shown in Figure 3.6. This is done because the Arduino operates on 5V and the Khadas on 3.3V.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);

  pinMode(SW_1      , INPUT);
  pinMode(SW_2      , INPUT);
  pinMode(SW_3      , INPUT);

  engine.init();
  stepper_1 = engine.stepperConnectToPin(Pul_1);
  stepper_2 = engine.stepperConnectToPin(Pul_2);
  stepper_3 = engine.stepperConnectToPin(Pul_3);
  stepper_1->setDirectionPin(Dir_1, false);
  stepper_2->setDirectionPin(Dir_2, false);
  stepper_3->setDirectionPin(Dir_3, false);

  stepper_1->setSpeedInHz(2600);       // 500 steps/s
  stepper_1->setAcceleration(10000);    // 100 steps/s²

  stepper_2->setSpeedInHz(2600);       // 500 steps/s
  stepper_2->setAcceleration(10000);    // 100 steps/s²

  stepper_3->setSpeedInHz(2600);       // 500 steps/s
  stepper_3->setAcceleration(10000);    // 100 steps/s²
  Calibrate();
}
```

Figure 3.10: Code excerpt of the stepper motor setup.

The motor angles are controlled by sending target positions to each motor. This target position is first converted from degrees to steps. This is done because the motors operate on steps, and not degrees. Figure 3.11 shows how the control is done.

```
    Target_1_s = int(Target_1*DegToStep);
    Target_2_s = int(Target_2*DegToStep);
    Target_3_s = int(Target_3*DegToStep);

    stepper_1->moveTo(Target_1_s);
    stepper_2->moveTo(Target_2_s);
    stepper_3->moveTo(Target_3_s);
```

Figure 3.11: Code of how the Arduino controls the stepper drives.

3.2.4 Communication

The communication between the Arduino and the Khadas runs on a serial protocol. This allows the Arduino to communicate with either the Khadas or a computer without having to do changes to the code. The communication is done, on the Arduino side, by using the function *Serial.readbytes()*, as seen in Figure 3.12.

The communication can either be done over a USB cable, or by connecting cables to the universal asynchronous receiver-transmitter ports on the Khadas and Arduino. The UART communication is set up, as seen in Figure 3.6, by connecting both RX/TX ports together. It is important to note that when an RX port is used on one card, it has to be connected to the TX port on the second card, and vice versa.

```
void UsbCom(){
  if (Serial.available() >= 16) { // Wait for 12 bytes (3 floats)

    Serial.readBytes((char*)values, 16);
    Target_1 = values[0];
    Target_2 = values[1];
    Target_3 = values[2];
    Calibrate_signal = values[3];
    Serial.print(Current_1_d);
    Serial.print(" ");
    Serial.print(Current_2_d);
    Serial.print(" ");
    Serial.print(Current_3_d);
    Serial.print(" ");
    Serial.println(Calibrate_signal);

  }
}
```

Figure 3.12: Code excerpt of the communication function in Arduino.

```
def USBArduinocom():

    PrevT = time.time()
    if platform.system() == 'Windows':
        Com = 'COM3'
    else:
        Com = '/dev/ttyS3'
    try:
        Arduino = serial.Serial(Com, 115200, timeout=1)
    except:
        C.Log.insert(0,'Failed Connection to Arduino')
        return
        pass

    while not C.exit_flag and C.Serial_Com_Running:

        CurrT = time.time()
        dt = CurrT - PrevT
        PrevT = CurrT
        if dt == 0:
            C.UartCom_Hz = 9999.99
        else:
            C.UartCom_Hz = 1 / dt
        # print(dt)


        if C.Paus_New_Arduino_Values:
            data = struct.pack('ffff', 0, 0, 0, 0)
        else:
            data = struct.pack('ffff', C.Stepper1_Target, C.Stepper2_Target, C.Stepper3_Target, float(C.Calibrate_Arduino))
        Arduino.write(data)
        response = Arduino.readline().decode().split()
        try:
            C.Stepper1_Feedback = float(response[0])
            C.Stepper2_Feedback = float(response[1])
            C.Stepper3_Feedback = float(response[2])
        except:
            pass
    C.UartCom_Hz = 0
```

Figure 3.13: Code excerpt of the communication function in Python.

The function used for the UART communication in Python is shown in Figure 3.13. First, the port used for the communication is selected based on which operating system the application is running on. Then a serial object is created from the *serialpy* library. This takes the communication port, baud rate, and timeout as arguments. Furthermore, in the while loop, a Byte package is created from the values to send. This byte package is then written to the Arduino card, followed by awaiting a response. This response is then written to its corresponding variables.

3.2.5 Limit Switches

Limit switches are added on every single motor. This is done to make sure that if the motors missed a step, they can be re-calibrated. When the limit switch is triggered, it sends a signal to the Arduino, and the code will start to run the motor back to the home position. This can be done by pressing the calibrate button on the touch display GUI. The limit switches are placed right before the lower bound angle of the motor leg. This is to ensure that leg can not collide with the motor bracket.

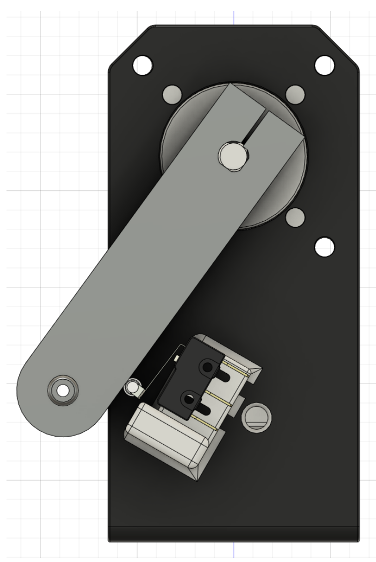


Figure 3.14: Limit switch triggered by the motor leg.

3.2.6 Calibration

The calibration sequence begins when the calibration function is called. The calibration sequence rotates the motors one position down until they trigger the limit switches. When a limit switch is triggered the corresponding motor stops turning. This sequence is shown in Figure 3.15

```
if (!sensorVal_1){
  stepper_1->move(-1);
}else{
  stepper_1->forceStop();
}

if (!sensorVal_2){
  stepper_2->move(-1);
}else{
  stepper_2->forceStop();
}

if (!sensorVal_3){
  stepper_3->move(-1);
}else{
  stepper_3->forceStop();
}
```

Figure 3.15: Code excerpt the limit switch trigger.

Once every limit switch has been triggered the motors update their current position to zero. This aligns all motors to have the same starting point. This sequence is shown in Figure 3.16. When all motors are calibrated, the platform returns to the starting position.

```
stepper_1->forceStopAndNewPosition(Pos_1_s);
stepper_2->forceStopAndNewPosition(Pos_2_s);
stepper_3->forceStopAndNewPosition(Pos_3_s);
Target_1_s = 0;
Target_2_s = 0;
Target_3_s = 0;
Calibrate_signal = -5.65;
```

Figure 3.16: Code excerpt of calibration.

3.2.7 Encoders

Encoders are mounted on the back end of the shaft from the stepper motors. These are used to measure the axial rotations of the shaft, to keep track of its position. Typically there are 2 types of encoders; absolute or incremental. The AMT102-V 3.2 that is used on this project, is a incremental encoder. The encoder sends feedback back to the Arduino. The signal travels through a special shielded cable to avoid any noise disrupting the signal. This is a common issue when mounting them close to the magnetic fields of a motor. The encoders are not used because stepper motor accuracy is sufficient on its own.

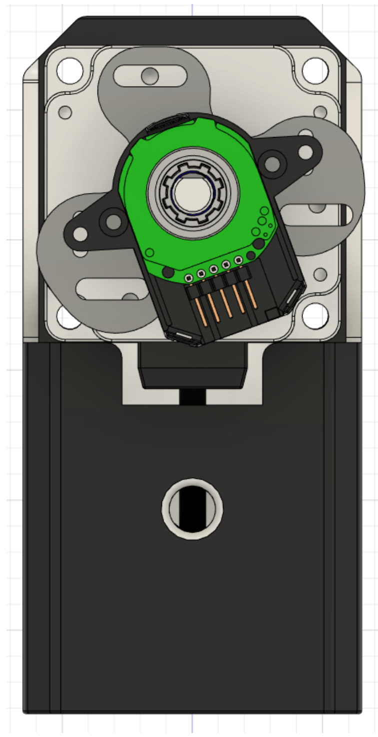


Figure 3.17: Encoder mounted on the stepper motor shaft.

3.2.8 Low-Pass Filter

A Low-pass filter is constructed to prevent issues with electromagnetic interference, also called noise. It is made out of capacitors and resistors from the lab. The components are soldered onto a PCB board in the same way as in the schematic 3.18. The schematic represents one filtered channel. The filter consists of 6 channels with room for more if needed. The limit switches are filtered before connecting to the Arduino. 3.6 This eliminates any noise being transmitted through the switches wires.

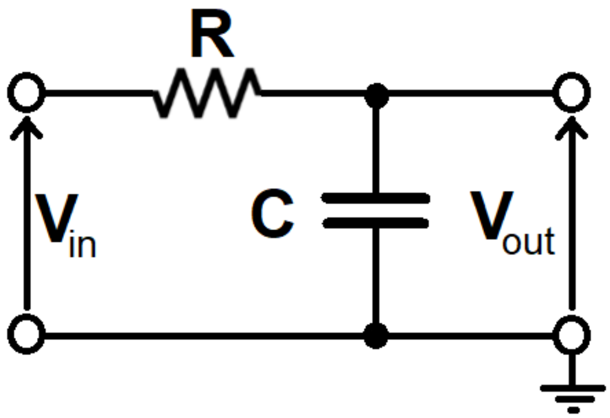


Figure 3.18: Electrical schematic of one filter channel.

3.2.9 Simulation-sketches

During the research and development stages, various sketches were drawn to demonstrate both the physical limits of the platform and the mathematical model of the system.

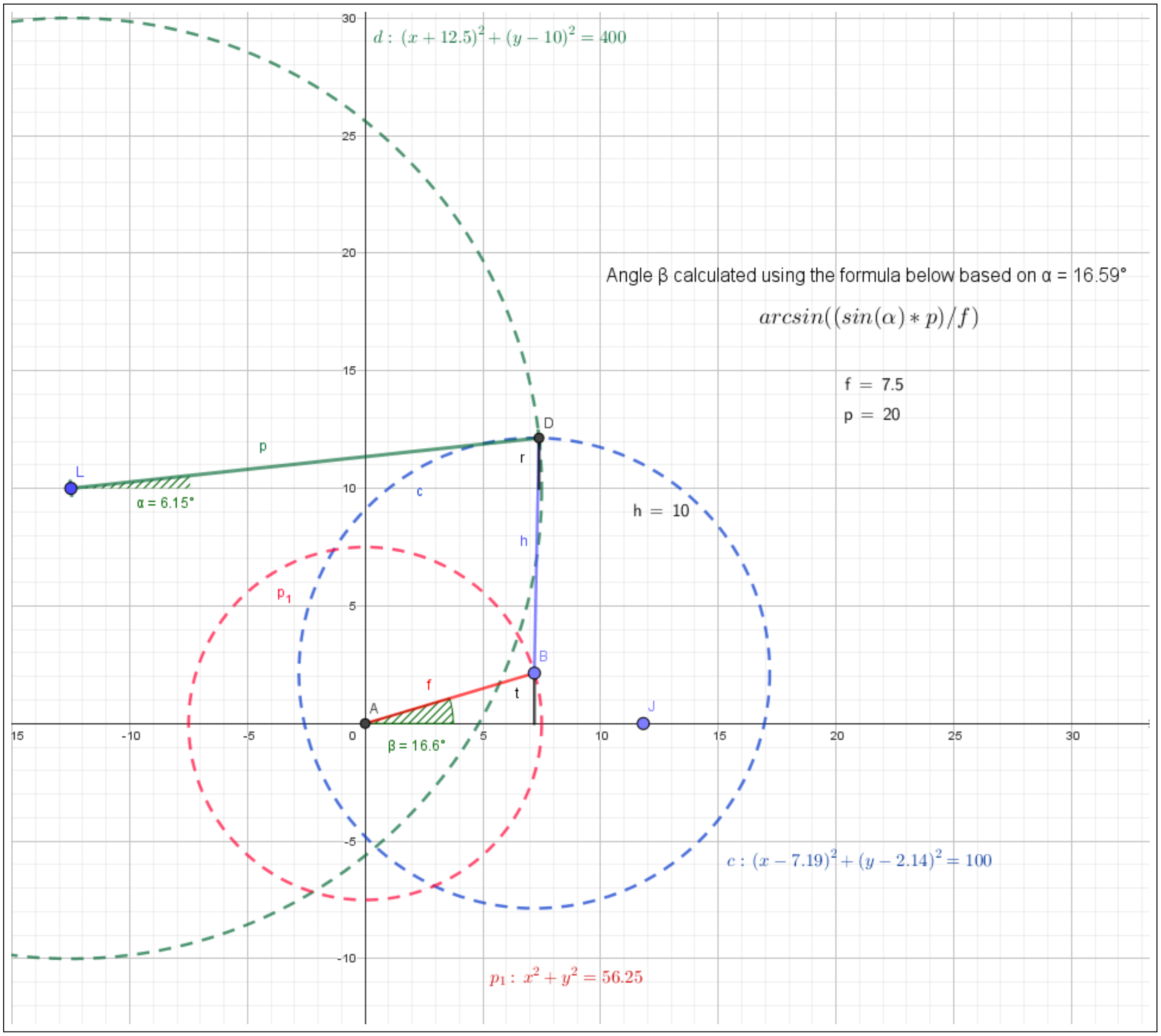


Figure 3.19: Geogebra simulation sketch of the correlation between platform angle β and motor angle α.

As you can see in Figure 3.19, the legs and angles of the platform are sketched. This is done to help find which angles for the platform and motors are to be used as physical limits. The Sketch is also drawn to suitable dimensions for the legs, and which formula is to be used to convert the platform angle to motor angle. The sketch is a simplified version and may not be accurate when working with three legs, or different configurations.

While sketching multiple good dimensions for the platform legs were found. In order to decide which dimensions were to be used, consistency and stability were taken to mind, and further testing had to be done to arrive at a decision. A motor leg length of 7.5 cm and a platform leg length of 10 cm proved to be a good combination to achieve good stability, robustness, size and functionality.

By sketching the dimension and movements of the leg, the physical limits for the platform angles were found to be ±18°. The formula to convert platform angle to motor angle was found by observing and analyzing the sketch. The formula was achieved by using simple trigonometry on the platform angle and leg dimensions. The formula is shown in equation 3.4

$$\arcsin(\sin\alpha \cdot p)/f = \beta \tag{3.4}$$

- α Is the platform angle.
- β Is the motor angle.
- **p** Is the length from the centre to the platform leg.
- **f** Is the length of the motor leg.

The formula for the motor angle α gives accurate conversions between the angles when close to zero, and only reaches an error of ±7% when the platform angle is at its maximum angle of ±20°. This error is seen as acceptable.

3.3 Camera

There exist many cameras suitable for object detection, the challenging part is to find a camera that fits the strict specifications of the design. The most important specification for a camera to be suitable for the design is that it has a field of view of a minimum of 100 degrees vertically and horizontally. One of the other specifications the camera needs to meet is small size. This is to ensure the camera can be mounted easily in the middle of the enclosure without obstructing other parts. Once these specifications are met there seem to be only two good camera options. The first option is the *Khadas 8MP HDR camera*[1]. The positive thing about this camera is that it comes from the same supplier as the Khadas VIM3 single-board computer. The other good camera option is the *Arducam 1080P Low Light WDR Ultra Wide Angle USB Camera*[2]. This camera

[1] https://www.khadas.com/product-page/os08a10-8mp-camera

[2] https://www.arducam.com/product/arducam-1080p-low-light-wdr-ultra-wide-angle-usb-camera-module-for-computer-2mp-cmos-imx291-160-degree-fisheye-mini-

Tech Specs.:

- Image Sensor: OmniVision's OS08A10
- Lens Size:1/2.0"
- Array Size:3840x2160
- Pixel Size:2.0 µm x 2.0 µm
- Megapixel:8 MegaPixel
- Interface:MIPI-CSI
- Lanes:4 lane
- Focal Length / EFL:3.47mm
- Aperture / F. No:f/2.1
- View Angle:160°
- Distortion:< -23%
- Focusing Range:80 ~ 500 cm
- Weight:13.2g
- Dimensions:100.0 * 26.0 * 20.6mm

(a) Khadas 8MP HDR camera.

Items	Parameters
Sensor	IMX291 SONY CMOS
Sensor Size	1/2.8"
Maximum Pixels	2.0 megapixel
Maximum Effective Resolution	1947(H)x1109(V)
Data Format	MJPG/YUY2/H.264
Pixel Size	2.9 µm x 2.9 µm
Dynamic Range	80 dB
Lens	FOV: 160° (D)
	Optical Format: 1/2.7"
	Lens construction: 6G+IR650
	F/NO: 2.0
Focusing	Fixed focus
IR Sensitivity	Integral IR filter, only visible light
Frame Rate	H.264 30fps@1920x1080; MJPG 30fps@1920x1080; YUY2 30fps@640x480
Auto Control	Saturation, Contrast, Acutance, White balance, Exposure
Adjustable Parameter	Brightness, Contrast, Saturation, Hue, Sharpness, Gamma, Gain, White Balance, Exposure
Audio	Single microphone (optional dual channel)
Input Voltage	DC 5V
Working Current	MAX 300mA
Operating Temp.	-4°F~158°F (-20°C~+70°C)
Cable Length	Default 1M, optional 2M, 3M, 5M
System Compatibility	Windows, Linux, Mac with UVC

(b) Arducam 1080P Wide Angle USB Camera.

is quite small and supports a wider range of lenses. The specifications for both of these cameras are shown in Figure 3.20a and 3.20b

After testing both of the cameras shown in Figure 3.20a and 3.20b, the most suitable choice seems to be the Arducam. This camera has a wider range of compatible lenses that can be attached, which allows the use of a wide-angle lens with appropriate FOV without too much distortion.

3.3.1 Ball Tracking Algorithm

This section describes the methods used to implement the ball-tracking algorithm. This includes the initialization and parameters for the camera, the algorithm to detect the ball, and the estimation of velocity.

---

uvc-usb2-0-spy-webcam-board-with-microphone-3-3ft-cable-for-windows-linux-mac-os//

3.3.2 Camera Initialization

The camera is initialized by using the OpenCV package in Python. OpenCV includes a wide range of user-friendly functions that make camera utilization easy. The camera is initialized by running the *VideoCapture* function, and based on which system is in use, this function communicates with the appropriate camera input. this is done by a simple IF-sentence that checks which operating system is in use. The video-feed resolution is set to a 4:3 format with 640 by 480 pixels. By using this resolution the pixels align correctly with the dimensions of the platform in millimeters. An excerpt of the initialization is shown in Figure 3.21.

```
# Initialize the video stream
if platform.system() == 'Linux':
    cap = cv2.VideoCapture(0)
elif platform.system() == 'Windows':
    cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)

# Set the video resolution to be 4:3, centered on (0,0)
width = 640
height = 480

cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
```

Figure 3.21: Code excerpt of camera initialization.

3.3.3 Colour Detection

To be able to accurately detect colours in the video feed, the frame is converted into HSV colour space. A threshold is then applied to only include a set of boundaries which are defined in HSV colour space. This boundary of colour representation is shown in the excerpt of code in Figure 3.23. By doing this the feed is only able to detect and view the colours which are defined by the boundary.

The next step is to obtain the coordinates of the colours which are included by the threshold. This is done by using the *findCountours* function included in OpenCV. This function searches for pixels which are visible in the feed. If the boundaries for the threshold are defined correctly, the only thing which is visible to the feed should be the colour of the ball. When the algorithm detects a contour in the feed, it draws a circle around it and returns the coordinates of the circle centre. To eliminate noise and unwanted pixels that show up in the feed, the algorithm only returns the coordinates for the maximum point of contours, and with a circle radius larger than 10 pixels. In the end, the coordinates of the ball are offset to have a point of origin in the centre of the frame. An excerpt of this algorithm is shown in Figure 3.22

```
# Convert the frame from BGR color space to HSV color space
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Threshold the image to isolate the orange color
mask = cv2.inRange(hsv, orange_lower, orange_upper)

# Find the contours in the mask
contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# If a contour is found, get its center and draw a circle around it
if len(contours) > 0:
    c = max(contours, key=cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
    if radius > 10:
        cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
        cv2.circle(frame, (int(x), int(y)), 2, (0, 255, 255), -1)
```

Figure 3.22: Code excerpt of camera colour detection.

```
# Define the lower and upper bounds of the orange color in HSV format
orange_lower = np.array([0, 88, 85])
orange_upper = np.array([13, 255, 201])
```

Figure 3.23: Code excerpt of HSV upper and lower bounds.

3.3.4 Velocity Estimation

The velocity estimation is done by deriving the position over time. To be able to estimate velocity, two different measurements of position are needed. These measurements are gathered from the colour detection and are assigned to a variable. An if-sentence then checks if a previous position measurement exists. If a previous measurement exists the derivative is calculated and assigned as the velocity estimate. If no other position measurement exists, the velocity is set as zero. After the velocity estimate is calculated, the current position gets assigned to a variable which is the previous position. This previous position carries over to the next iteration and is used to calculate the next velocity estimate. A code excerpt of this estimation is shown in Figure

3.24.

```
if gx_prev == gx:
    gx_vel = 0
else:
    gx_vel = (gx - gx_prev) / dt
if gy_prev == gy:
    gy_vel = 0
else:
    gy_vel = (gy - gy_prev) / dt

gx_prev = gx
gy_prev = gy
```

Figure 3.24: Code excerpt of velocity estimation.

3.4 Control Algorithm

This section describes the development and methods used to implement the MPC control algorithm for the platform. The methods include the generation of C code, problem formulation, methods to reduce time, and the main algorithm for the controller.

3.4.1 MPC Mathematical Model

The models and formulas in equations 3.5 - 3.12 are derived from the article [17] and [9]. The models and formulas are altered to fit the 3DOF motion platform system.

Given that air resistance, friction, centrifugal force and other small forces of error and noise are neglected. The nonlinear relation between the platform and the ball along the X-axis becomes:

$$a = \frac{5}{7}(g \sin\phi - \dot{\phi}^2 l) \tag{3.5}$$

Because the platform is constrained between an angle of ±0.43 radians, $\sin\phi$ and $\sin\theta$ can be approximated to ϕ and θ. This removes the non-linearity and the relation between the platform and the ball becomes equation 3.6. The units in the equation are also adjusted from meters to millimetres explaining the presence of the number 1000, which is a conversion factor.

$$
\begin{aligned}
a &= \frac{5}{7} \cdot g \cdot 1000 \\
\ddot{x}_b(t) &= \phi a(t) \\
\ddot{y}_b(t) &= \theta a(t)
\end{aligned}
\tag{3.6}
$$

The equation above can be represented as:

$$
\dot{x}(t) = A_c x(t) + B_c u(t) \tag{3.7}
$$

The state vector **x** and input vector **u** is given as:

$$
\mathbf{x} = \begin{bmatrix} x_b \\ \dot{x}_b \\ y_b \\ \dot{y}_b \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \phi \\ \theta \end{bmatrix} \tag{3.8}
$$

The continuous-time state space model is given as:

$$
\mathbf{A_{continuous}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{B_{continuous}} = \begin{bmatrix} 0 & 0 \\ a & 0 \\ 0 & 0 \\ 0 & a \end{bmatrix} \tag{3.9}
$$

The discrete-time state space model is Euler discretized by:

$$
\begin{aligned}
A &= A_c \Delta t + I \\
B &= B_c \Delta t
\end{aligned}
\tag{3.10}
$$

Which gives the discrete time state space model as:

$$
x(k+1) = Ax(k) + Bu(k) \tag{3.11}
$$

$$
\mathbf{A} = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ a\Delta t & 0 \\ 0 & 0 \\ 0 & a\Delta t \end{bmatrix} \tag{3.12}
$$

The MPC problem formulation is formulated as a cost function of the sum of squares of the error in the states + the input control angle. The angles are constrained between ±0.43 radians. The states are updated for each time step by using the discretized model matrices in equation 3.12.

$$
\begin{aligned}
\text{Minimize:} \quad & \mathbf{J} = \sum_{i=0}^{20-1} \left[(\mathbf{x}(i) - \mathbf{x}_{ref})^T \mathbf{Q} (\mathbf{x}(i) - \mathbf{x}_{ref}) + \mathbf{u}(i)^T \mathbf{R} \mathbf{u}(i) \right] \\
\text{Subject to:} \quad & \mathbf{x}(\mathbf{0}) = \mathbf{x}_{initial} \\
& \mathbf{x}(i+1) = \mathbf{x}(i) + (\mathbf{A}\mathbf{x}(i) + \mathbf{B}\mathbf{u}(i)), \quad i = 0 \dots 20-1 \\
& \mathbf{-0.43} \le \mathbf{u}(i) \le \mathbf{0.43}
\end{aligned}
\tag{3.13}
$$

$\mathbf{x}(i)$ is the system state at time step i.

$\mathbf{u}(i)$ is the control input at time step i.

$\mathbf{x}_{initial}$ is the initial conditions for the states.

$\mathbf{x}_{ref}$ is the reference states.

$\mathbf{Q}$ and $\mathbf{R}$ are positive definite weighting matrices.

$\mathbf{A}$ and $\mathbf{B}$ are the discrete state and input matrices of the system model.

3.4.2 Generating C Code

The MPC problem formulation is created in C and later used in Python. This is done to decrease the amount of time the algorithm uses to calculate the optimal angle. The problem is formulated in Jupyter Notebook once by using the CVXPY and CVXPYgen packages. CVXPY also makes use of the DCP and DPP concepts and rulesets to further decrease the time it takes for the algorithm to calculate the optimal angle. Figure 3.25 contains a code excerpt which defines the variables and parameters in the problem. Figure 3.26 contains the problem formulation as a for loop to calculate the desired angle over the time horizon. These code excerpts are the main necessary components required to generate the MPC problem in C. The technicalities of converting written Python code to C code and wrapping this code are done by the CVXPYgen package. This simplifies the process and gives extra time to focus on the MPC problem formulation itself. The C code must be generated on the system it is planned to be used on.

```
# define dimensions
H, n, m = 20, 4, 2

# define variables
U = cp.Variable((m, H), name='U')
S = cp.Variable((n, H+1), name='S')

# define parameters
Q = cp.Parameter((n, n), name='Qsqrt')
R = cp.Parameter((m, m), name='Rsqrt')
A = cp.Parameter((n, n), name='A')
B = cp.Parameter((n, m), name='B')
s_error = cp.Parameter((n, 1), name='s_error')
dt = 0.05
```

Figure 3.25: Code excerpt of C code generation of problem parameters.

The code allows for easy tuning and edits in the problem parameters when writing as shown in Figure 3.25. By doing this the parameters can be edited in Python without needing to generate the code from scratch or lowering iteration time.

```
# define objective
for t in range(H):
    #Make the Cost Function in terms of total error from reference point + control angle
    cost += cp.sum_squares(Q@(S[:, t:t+1]))
    cost += cp.sum_squares(R@(U[:, t]))
    #Update position and velocity states for the next timestep
    constr.append(S[:, t+1] ==  S[:, t] + dt*(A @ ((S[:, t])) + B@U[:, t]))
    #Constrain the control angle in radians
    constr += [U[:, t] <= 0.43]
    constr += [U[:, t] >= -0.43]
constr += [S[:, 0] == s_error[:, 0]]

#Solve the problem based on the optimal trajectory and input angle from the MPC
problem = cp.Problem(cp.Minimize(cost), constr)
```

Figure 3.26: Code excerpt of C code generation of problem formulation.

In Figure 3.26 The cost function of the MPC problem is written as the cost to be minimized in the problem. The constraints of the problem are defined in the *constr* list. This list is updated for each iteration of the for loop.

3.4.3 MPC Algorithm

The main MPC used to calculate the optimal control angle consists of two main parts, but also includes miscellaneous code which acquires and calculates error. The first part consists of defining parameters and tuning values used in the MPC. This includes both the A and B model matrices, and the Q and R weighting matrices. The code used to define these is shown in Figure 3.27. Because of the methods used to define these parameters in the generation of C code, these parameters can easily be tuned in the main script without generating new C code for each change.

```
# Assign Parameters for MPC
Apar = np.array([[0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 1.0], [0.0, 0.0, 0.0, 0.0]])
Bpar = np.array([[0.0, 0.0], [7000.0, 0.0], [0.0, 0.0], [0.0, 7000.0]])

problem.param_dict['A'].value = Apar
problem.param_dict['B'].value = Bpar

lock.acquire()
MPC_Q_xp = shm_array[15]
MPC_Q_xv = shm_array[16]
MPC_R_Vin = shm_array[17]
lock.release()

MPC_q = np.diag([MPC_Q_xp, MPC_Q_xv, MPC_Q_xp, MPC_Q_xv])
MPC_r = np.diag([MPC_R_Vin, MPC_R_Vin])

problem.param_dict['Qsqrt'].value = MPC_q
problem.param_dict['Rsqrt'].value = MPC_r
```

Figure 3.27: Code excerpt from the main algorithm of lines defining MPC parameters.

The main loop running the MPC controller is shown in Figure 3.28. This excerpt takes the state errors and inserts them into the imported C code of the problem formulation. The C code then runs the problem and extracts the optimal X and Y angles which should be used to control the platform.

```
# Update States
s_states = np.array([[x_pos], [x_vel], [y_pos], [y_vel]])
s_error = (s_r - s_states)# Initial state

# Solve MPC Problem
problem.param_dict['s_error'].value = s_error
problem.register_solve('CPG', cpg_solve)
problem.solve(method='CPG')
angle_list = problem.var_dict['U'].value
state_list = problem.var_dict['S'].value
u_traj = angle_list[:, 1]
```

Figure 3.28: Code excerpt from the main algorithm with the lines running the MPC controller.

3.4.4 Additional Control Algorithms

To be able to compare and check the results of the MPC two additional controllers are made. These controllers act as a reference for the MPC to be able to confirm that the MPC works as intended. These two controllers are a proportional–integral–derivative controller and a state space controller. Both controllers are derived from the report [15] and changed to fit the current system.

PID

A simple PID controller was implemented for comparison. The code for the PID controller is shown in Figure 3.29. The PID calculates the appropriate angle by taking the sum of each PID term multiplied by the error in the system. More information regarding the PID controller can be found in the report [15].

```
#------------------------------------------------------------- Pid x
Error_x = Feedback_x_p - Target_x_p

DeDt_x = (Error_x-PrevError_x)/dt
integral_x = integral_x + Error_x*dt

PrevError_x = Error_x

U_x = Pid_p_x*Error_x + Pid_d_x*DeDt_x + Pid_i_x*integral_x
#------------------------------------------------------------- Pid y
Error_y = Feedback_y_p - Target_y_p

DeDt_y = (Error_y - PrevError_y) / dt
integral_y = integral_y + Error_y * dt
PrevError_y = Error_y

U_y = Pid_p_y * Error_y + Pid_d_x * DeDt_x + Pid_i_y*integral_y


PrevError_y = Error_y
```

Figure 3.29: Code excerpt of the PID Controller.

StateSpace

A simple State Space controller was implemented for comparison. The state space controller is shown in Figure 3.30. The controller calculates the appropriate control angle by multiplying the control gains by the error in the system. More information about the controller gains, and the mathematical equations, can be found in the report [15].

```
x_p_Error =  Feedback_x_p - Target_x_p
U_x = (x_p_Error * SS_k1_x + x_v_Error * SS_k2_x)

y_p_Error =  Feedback_y_p - Target_y_p
U_y = (y_p_Error * SS_k1_y + y_v_Error * SS_k2_y)

PrevFeedback_x_p = Feedback_x_p
PrevFeedback_y_p = Feedback_y_p
```

Figure 3.30: Code excerpt of the state space controller.

3.4.5 Motor Angles

The controllers only calculate the necessary platform angle on the X and Y-axis. Because of this, the platform angle has to be converted to the appropriate motor angle for each of the three motors. The necessary equations required to do this are derived from Section 2.3.3 in the article [9]. These equations are combined with Equation (3.4) to give the mathematical operations shown in Figure 3.31.

```
#-------------------------------------------------------------

# l = 426.5mm - 123.12, 213,25, (sq(3)*l)/6, l/2
# l2 = 346.41mm, 99.99, 173.205

if C.Running:
    #
    t1 = 100*np.sin(C.U_rad_y)*np.cos(C.U_rad_x)
    t2 = 173.2*np.sin(C.U_rad_x)

    z1 = -t1 - t2
    z2 = -t1 + t2
    z3 =  t1

    z1 = LimitAngle(z1)
    z2 = LimitAngle(z2)
    z3 = LimitAngle(z3)

    C.Stepper1_Target = np.arcsin(z3/75)/RtD
    C.Stepper2_Target = np.arcsin(z2/75)/RtD
    C.Stepper3_Target = np.arcsin(z1/75)/RtD

sin_y_cos_x = (75 * np.sin(C.Stepper1_Feedback * RtD)) / 100
sin_x = ((75 * np.sin(C.Stepper2_Feedback * RtD)) + 100 * sin_y_cos_x) / 173.2
C.U_x_v_fb = round(np.arcsin(sin_x) / RtD, 2)

c = sin_y_cos_x/np.cos(np.arcsin(sin_x))
if c > 1:
    c = 1
elif c < -1:
    c = -1
C.U_y_v_fb = round((np.arcsin(c))/RtD, 2)
time.sleep(0.005)
```

Figure 3.31: Code excerpt of the angle conversions.

3.5 Graphical User Interface

This section describes the methods used to program the GUI. The GUI is written in Python using the PyQT5 library.

3.5.1 Controller Config

The controller config consists of a config window frame. This frame makes it possible to change the parameters for the controller through a user interface. The config frames are implemented by creating a QWidget class which describes the layout of the controller config UI. This class is created for each config, such that all configs follow the same structure.

```python
class MPC_Config(QWidget):
    def __init__(self):
        super().__init__()
        self.Decimal = None
        self.setGeometry(700,350,570,440)
        self.setWindowTitle('MPC Config')

        self.font = QFont()
        self.font.setPointSize(16)

        self.font2 = QFont()
        self.font2.setPointSize(12)
        #---------------------------------------------------------- info
        self.Info = QLabel(self)
        self.Info.move(10,5)
        self.Info.setText('Change the Q and R matrix. Save and rerun the MPC')
        self.Info.setFont(self.font)

        self.Info2 = QLabel(self)
        self.Info2.move(10, 335)
        self.Info2.setText('NB! all values are multiplied by 10')
        self.Info2.setFont(self.font2)
```

Figure 3.32: Code excerpt of the config frame for the MPC.

In Figure 3.32 a class named MP_Config is defined, which inherits from the QWidget class. By using the QWidget class, one can create frame elements by calling the objects from the class. The frame's geometry is set and information labels are created and placed in the frame. The font object is used to set the text size.

```
#---------------------------------------------------------- Set K1 value
self.LableK1 = QLabel(self)
self.LableK1.move(65, 45)
self.LableK1.setText('Q Position')
self.LableK1.setFont(self.font)

self.k1Gain = QDoubleSpinBox(self)
self.k1Gain.setObjectName(u"doubleSpinBox")
self.k1Gain.setGeometry(QRect(5, 80, 220, 100))
self.k1Gain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                          "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
self.k1Gain.setDecimals(4)
self.k1Gain.setMinimum(0.00000)
self.k1Gain.setSingleStep(0.01000)
```

Figure 3.33: Code excerpt of the gain button definition.

In Figure 3.33 a QDoubleSpinBox object is created. This object allows the user to set its value by using two buttons on the right side of the object. In the setStyleSheet, the button dimensions are enlarged. Two more QDoubleSpinnBox objects are also created for the Q Velocity and R value, these follow the same setup.

```
#---------------------------------------------------------- Set Decimal
self.LableDesi = QLabel(self)
self.LableDesi.move(300, 190)
self.LableDesi.setText('Number of decimals')
self.LableDesi.setFont(self.font)

self.Decimal = QSpinBox(self)
self.Decimal.setObjectName(u"Decimal")
self.Decimal.setGeometry(QRect(300, 220, 220, 100))
self.Decimal.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                           "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
self.Decimal.setMinimum(-5)
self.Decimal.setMaximum(0)
self.Decimal.setSingleStep(1)
self.Decimal.setValue(-2)
self.Decimal.valueChanged.connect(self.updateDecimal)
```

Figure 3.34: Code excerpt of the decimal shift button.

```
def updateDecimal(self):
    self.k1Gain.setSingleStep(10 ** (self.Decimal.value()))
    self.k2Gain.setSingleStep(10 ** (self.Decimal.value()))
    self.k3Gain.setSingleStep(10 ** (self.Decimal.value()))
```

Figure 3.35: Code excerpt of the decimal update function.

In the QDoubleSpinBox from Figure 3.34, a setSingleStep is used to set how much the value will change on each click of the button. To make it easy to use, a new QSpinBox object is created to change the step value by a factor of 10^X where X goes from -5 to 0. This is done by connecting the *Decimal* object to trigger the *updateDecimal* function when the value is changed. The *updateDecimal* function is shown in Figure 3.35.

```
        #------------------------------------------------- Save and close
        self.SaveButton = QPushButton("Save", self)
        self.SaveButton.setGeometry(287, 355, 283, 80)
        self.SaveButton.clicked.connect(self.Save)
        # ------------------------------------------------Cancel
        self.CancelButton = QPushButton("Cancel", self)
        self.CancelButton.setGeometry(2, 355, 275, 80)
        self.CancelButton.clicked.connect(self.Cancel)
```

Figure 3.36: Code excerpt of the save and Cancel objects.

```
    def Cancel(self):
        self.hide()

    def Save(self):
        Config.Log.append('New values saved for the MPC')
        Config.MPC_Q_xp = round(self.k1Gain.value()*10, 5)
        Config.MPC_Q_xv = round(self.k2Gain.value()*10, 5)
        Config.MPC_R_Vin = round(self.k3Gain.value()*10, 5)
        self.hide()
```

Figure 3.37: Code excerpt of the save and cancel function.

To save the new values a QPushButton object is created. This object is connected to the *Save* function when clicked. This *save* function is shown in Figure 3.37. This Function updates the internal variable in the Config.py file to the corresponding QDoubleSpinnBox object value and then hides the frame. To cancel or abort the changes to the new values a *Cancel* QPushButton object is created to hide the frame when clicked. The cancel and save objects are shown in Figure 3.36.

3.5.2 Main GUI

The main GUI code contains numerous repetitive objects, for the sake of simplicity, the various objects will be represented in lists.

```python
class GUI(QMainWindow):
    def __init__(self):
        super(GUI, self).__init__()

        self.PID_config = PID_Config()
        self.ss_config = SS_Config()
        self.MPC_config = MPC_Config()

        self.setWindowTitle("3 DOF")
        self.setGeometry(0,0, 1280, 800)

        self.circle_rect = QRect(QPoint(50, 80), QPoint(750, 750))
        self.ball_center = None
        self.FB_ball = None
        self.log_string = []
        self.log_string_old = []
        self.font1 = QFont()
        self.font1.setPointSize(9)
```

Figure 3.38: Code excerpt of the main GUI frame class.

The main GUI class shown in Figure 3.38 starts by getting an instance of the controller config classes. This is done so that the main GUI

Labels

Figure 3.4 contains all the QLabel objects created for the GUI class in the GUI.py script. These objects are used for two different cases described by the comment in Figure 3.4. *Text label* defines a static label used for informative text. *Value label* is used to represent a label where the text changes over time.

Nr	QLabel	Line nr	Comment
1	CPUTmpLabel	341	Text label
2	CPUTmp1	344	Value label
3	CPUTmp2	349	Value label
4	ControlModeLable	360	Text label
5	CameraBoxLable	406	Text label
6	ArduinoBoxLable	449	Text label
7	Stepper1_lable	487	Text label
8	Stepper1_TargetValue	491	Value label
9	Stepper1_FeedbackValue	496	Value label
10	Stepper2_lable	501	Text label
11	Stepper2_TargetValue	505	Value label
12	Stepper2_FeedbackValue	510	Value label
13	Stepper3_lable	515	Text label
14	Stepper3_TargetValue	519	Value label
15	Stepper3_FeedbackValue	524	Value label
16	TargetBoxLable	531	Text label
17	PositionX	540	Value label
18	PositionY	544	Value label
19	VelX	548	Value label
20	VelY	552	Value label
21	AngleX	556	Value label
22	AngleY	560	Value label
23	TargetValues	566	Text label
24	TargetPosX	570	Value label
25	TargetPosY	573	Value label
26	TargetVelocityY	576	Value label
27	TargetVelocityX	579	Value label
28	TargetAngleX	582	Value label
29	TargetAngleY	585	Value label
30	FeedbackValues	590	Text label
31	FeedbackPosX	594	Value label
32	FeedbackPosY	597	Value label
33	FeedbackVelocityY	600	Value label
34	FeedbackVelocityX	603	Value label
35	FeedbackAngleX	606	Value label
36	FeedbackAngleY	609	Value label
37	ErrorValues	614	Text label
38	ErrorPosX	618	Value label
39	ErrorPosY	621	Value label
40	ErrorVelocityY	624	Value label
41	ErrorVelocityX	627	Value label
42	ErrorAngleX	630	Value label
43	ErrorAngleY	633	Value label
44	HzLable1	642	Text label
45	GUIHz	645	Value label
46	HzLable2	648	Text label
47	SharedMemHz	651	Value label
48	HzLable3	654	Text label
49	BackEndHz	657	Value label
50	HzLable4	660	Text label
51	UartComHz	663	Value label
52	HzLable5	666	Text label
53	CameraHz	669	Value label
54	HzLable6	672	Text label
55	ControllerHz	675	Value label
56	LogBoxLable	680	Text label

Table 3.4: Table of QLabel objects from GUI.py.

Functions

To execute background tasks functions need to be created. These functions perform tasks when executed by an object. Most of the tasks change a value from True to False. All of these functions created is described in Table 3.5. Some of the functions have specific tasks and are explained further below.

Functions	Line nr	Comment
on_button_clicked	689	Changes the button colour when clicked
UpdateInformativeValues	698	Updates all value lables
StartValuesRecord	773	Sets Record_Data value high or low in Config.py
PauseAndSetArduinoValues	779	Sets Pause_New_Arduino_Values value high or low in Config.py
StartButton1	785	Sets Start value high or low in Config.py
CameraStart	793	Sets Camera_Start value high or low in Config.py
CameraShowFrame	799	Sets Camera_show_Frame value high or low in Config.py
CameraShowMask	805	Sets Camera_Show_Mask value high or low in Config.py
PauseAndSetCameraValues	811	Sets Camera_Pause value high or low in Config.py
StartSerialCom	819	Sets serial start signal to high or low in Config.py
StartI2CCom	826	Not used
CalibrateArduino_True	832	Sets calibration value high
CalibrateArduino_False	837	Sets calibration value low
MPC_Snap_Shot_True	840	Sets snapshot value high
MPC_Snap_Shot_False	846	Sets snapshot value low
ControllerSelecter	850	passes the controller name to config.py
InputtModeSelector	854	Passes the mode selected to config.py
paintEvent	857	Paints the target ball and feedback ball
mousePressEvent	874	Updates the targe position from click action
mouseMoveEvent	882	Updates the targe position from drag motion
EditConfig	892	Not used
toggle_window	897	Opens the Controller config frame
CloseButton	921	Close application confirm popup
closeEvent	929	Closes the application

Table 3.5: Table listing all functions.

```
689 def on_button_clicked(self):
690     sender = self.sender()
691     if sender.isChecked():
692         sender.setStyleSheet("background-color: green")
693     else:
694         sender.setStyleSheet("background-color: red")
```

Figure 3.39: Colour changing function.

The function shown in Figure 3.39 takes the QPushButton object that executes it as an input argument. This function then changes the object's colour to red or green, dependent on if it is checked or not.

```
857     def paintEvent(self, event):
858         painter = QPainter(self)
859         pen = QPen(Qt.black, 2, Qt.SolidLine)
860         brush = QColor(0, 0, 0, 0)
861         painter.setPen(pen)
862         painter.setBrush(brush)
863         painter.drawEllipse(self.circle_rect)
864         if self.Target_ball_pos:
865 
866             brush1 = QColor(255, 0, 0)
867             painter.setBrush(brush1)
868             painter.drawEllipse(self.Target_ball_pos, 10, 10)
869             brush2 = QColor(255, 255, 0)
870             painter.setBrush(brush2)
871             painter.drawEllipse(self.FB_ball, 5, 5)
872 
873     def mousePressEvent(self, event):
874         if event.button() == Qt.LeftButton:
875             if self.circle_rect.contains(event.pos()):
876                 Config.Target_Pos_x = round((400/700)*(-400 + event.x()), 1)
877                 Config.Target_Pos_y = round((400/700)*(400 - event.y()), 1)
878                 self.Target_ball_pos = event.pos()
879                 self.update()
880 
881     def mouseMoveEvent(self, event):
882         if self.circle_rect.contains(event.pos()):
883             Config.Target_Pos_x = round((400/700)*(-400 + event.x()), 1)
884             Config.Target_Pos_y = round((400/700)*(400 - event.y()), 1)
885             self.Target_ball_pos = event.pos()
886             self.update()
```

Figure 3.40: Paint functions.

To paint a circle to represent the physical platform on the GUI, a QPainter object is used. This object can paint shapes to the GUI. In the *paintEvent* function on line 867 from Figure 3.40, the physical platform border, and the two circles representing the target and feedback position are painted. They are painted by *painter.drawElipse()* where the pixel coordinates, width, and height is given as the argument.

There are two more functions named *mousePressEvent* and *mouseMoveEvent* from Figure 3.40. These two functions trigger whenever an event happens, and then take the event as an argument. The purpose of these two functions is to check if a left mouse button or a click-drag event has happened inside the platform's circle. If an event happens inside of the border, the pixel coordinate of that event is written to the *Target_ball_pos* variable. These pixel coordinates are offset to the middle of the platform circle, converted to millimetres, and stored to their representative variable in Config.py.

```python
def toggle_window(self):
    if Config.Control_Mode == 'PID':
        if not self.PID_config.isVisible():
            self.PID_config.setWindowFlags(Qt.WindowStaysOnTopHint)
            self.PID_config.show()
            self.PID_config.PGain.setValue(Config.PID_P_x/10)
            self.PID_config.DGain.setValue(Config.PID_D_x/10)
            self.PID_config.IGain.setValue(Config.PID_I_x/10)

    if Config.Control_Mode == 'StateSpace':
        if not self.ss_config.isVisible():
            self.ss_config.setWindowFlags(Qt.WindowStaysOnTopHint)
            self.ss_config.show()
            self.ss_config.k1Gain.setValue(Config.SS_k1_x/10)
            self.ss_config.k2Gain.setValue(Config.SS_k2_x/10)

    if Config.Control_Mode == 'MPC':
        if not self.MPC_config.isVisible():
            self.MPC_config.setWindowFlags(Qt.WindowStaysOnTopHint)
            self.MPC_config.show()
            self.MPC_config.k1Gain.setValue(Config.MPC_Q_xp/10)
            self.MPC_config.k2Gain.setValue(Config.MPC_Q_xv/10)
            self.MPC_config.k3Gain.setValue(Config.MPC_R_Vin/10)
```

Figure 3.41: Show controller config frame.

To display a controller config window frame, it has to be called by *.show()*. The function in Figure 3.41 checks which controller is selected, and shows the config frame for that controller. In addition to this, the function updates the values of the QDoubleSpinBox objects in the config class to be shown. To make sure the frame doesn't disappear behind the main application frame, the *.setWindowFlags(Qt.WindowStaysOnTopHint)* is used to force the frame to be on top.

```python
def CloseButton(self):
    # Display a confirmation dialog before quitting the application
    reply = QMessageBox.question(self, 'Confirm Exit', 'Are you sure you want to exit?',
                                 QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

    if reply == QMessageBox.Yes:
        QApplication.closeAllWindows()

def closeEvent(self, event):
    # Set a flag to signal the threads to exit
    QApplication.closeAllWindows()
    Config.exit_flag = True
```

Figure 3.42: Close and exit function.

For the application to close properly, all frames and background tasks need to be closed. This task is done by the function *CloseButton* in Figure 3.42. This function displays a popup window where you have to confirm if you want to exit. This is done so that a miss click won't happen. If you answer yes all frames that are open will close, and a close event will be sent. This close event triggers the *closeEvent* function in Figure 3.42 and sets the application exit_flag to True. Two functions are used for this because a close event can also come from pressing the close/exit button located in the top right corner.

Buttons

For interaction with the GUI, QPushButton objects are used. These objects can execute functions from different actions by adding *.clicked*, *.pressed* or *.released* followed by *.connect()* to connect it to a function when that action happens.

QPushButton	Checkable	Connect		Line nr	Comment
		Type	Target function		
ExitButton	FALSE	clicked	CloseButton	335	Button to close the application
EditButton	FALSE	clicked	toggle_window	374	Button to show the config frame
StartButton	TRUE	clicked	StartButton1	388	Button to start the controller
		clicked	on_button_clicked		
RecordData	TRUE	clicked	on_button_clicked	396	Button to start the recording
		clicked	StartValuesRecord		
		pressed	MPC_Snap_Shot_True		
		released	MPC_Snap_Shot_False		
StartCamera	TRUE	clicked	on_button_clicked	416	Button to start the camera code
		clicked	CameraStart		
CamerashowFrame	TRUE	clicked	on_button_clicked	424	Button to show camera feed
		clicked	CameraShowFrame		
CamerashowMask	TRUE	clicked	on_button_clicked	431	Button to show camera mask
		clicked	CameraShowMask		
CameraStopReading	TRUE	clicked	on_button_clicked	438	Button to not use the camera values
		clicked	PauseAndSetCameraValues		
SerialConnect	TRUE	clicked	on_button_clicked	459	Button to start the serial communication
		clicked	StartSerialCom		
PausReadWriteConnect	TRUE	clicked	on_button_clicked	467	Button to puase the sending of values
		clicked	PauseAndSetArduinoValues		
Calibrate	FALSE	pressed	CalibrateArduino_True	478	Button to start the arduino internal calibration
		released	CalibrateArduino_False		

Table 3.6: Table listing all buttons.

In Table 3.6 all the QPushbutton objects are shown. The Table also lists which function they are connected to by what action triggers it.

```python
# ---------------------------------------------------------------------- Record data
self.RecordData = QPushButton('Record Data', self)
self.RecordData.setGeometry(1130, 110, 100, 50)
self.RecordData.setCheckable(True)
self.RecordData.setStyleSheet("background-color: red")
self.RecordData.clicked.connect(self.on_button_clicked)
self.RecordData.clicked.connect(self.StartValuesRecord)
self.RecordData.pressed.connect(self.MPC_Snap_Shot_True)
self.RecordData.released.connect(self.MPC_Snap_Shot_False)
```

Figure 3.43: Record Data button.

From Figure 3.43 the Record Data button object declaration is shown. This shows the basic structure of a QPushButton object declaration, and how it is connected to functions.

3.5.3 Controller And Input Mode Selector

The application contains three different controllers. In order to change which controller is selected, a QComboBox object is created. This object is shown in Table 3.7 and Figure 3.44, and the function is shown in Figure 3.45.

QComboBox	Selection option	Connect	Line nr	Comment
Controller	No Control Mode selected	ControllerSelecter	364	Box where it is possible to select controller
	PID			
	StateSpace			
	MPC			
InputMode	No Input selected	InputtModeSelector	379	Box where it is possible to select imput mode (Not used)
	Hunt (not implemented)			
	Track (not implemented)			

Table 3.7: QComboBox objects.

```python
        #------------------------------------------------------------------ Controller selector
        self.ControlModeLable = QLabel(self)
        self.ControlModeLable.setGeometry(660, 85, 180, 20)
        self.ControlModeLable.setText("Controller settings: ")

        self.Controller = QComboBox(self)
        self.Controller.setFont(self.font1)
        self.Controller.addItem("No Control Mode selected")
        self.Controller.addItem("PID")
        self.Controller.addItem("StateSpace")
        self.Controller.addItem("MPC")
        self.Controller.setObjectName(u"comboBox1")
        self.Controller.setGeometry(QRect(670, 110, 190, 50))
        self.Controller.activated[str].connect(self.ControllerSelecter)
```

Figure 3.44: Code excerpt of the controller selector.

```python
    # ------------------------------------------------------------------ Write Control Mode to config file
    def ControllerSelecter(self, text):
        Config.Control_Mode = text
```

Figure 3.45: Code excerpt of the controller selector function.

From Table 3.7, two QComboBox objects are created, but only the Controller object is used further in the application. This object is the QComboBox object shown in Figure 3.44. This object has a dropdown menu showing all the items it contains. All the controllers are added as string variables. Whenever a new item is selected from the drop-down menu, a string variable will be passed to the function *ControllerSelector* shown in Figure 3.45. This function takes the item's string value as an argument and stores it in the Config.py file.

Miscellaneous objects

This section describes miscellaneous objects used in the GUI class.

QFrame	Line nr	Comment
ControlerBox	354	Creat a box arount buttons
CameraBox	410	Creat a box arount buttons
ArduinoBox	453	Creat a box arount buttons
TargetBox	535	Creat a box arount values
Hzbox	637	Creat a box arount values
QTextEdit	Line	Comment
LogBox	684	Text box for log tex
QMessageBox	Line	Comment
reply	924	pop up window fro confirmation

Table 3.8: Table listing miscellaneous objects.

Table 3.8 shows a list of the different objects used for special purposes. The *QFrame* object is used to create a box frame around buttons and labels to get a graphical collection. The *QTextBox* object is used to create a text frame where a string array can be shown. The *QMessageBox* objects create a popup window that returns an action.

Update GUI values

```
# ----------------------------------------------------------------- Update Informative Values
def UpdateInformativeValues(self):
    self.Stepper1_TargetValue.setText('Target Angle      :   '+ str(round(Config.Stepper1_Target,3)))
    self.Stepper1_FeedbackValue.setText('Feedback Angle :   '+ str(round(Config.Stepper1_Feedback,3)))

    self.Stepper2_TargetValue.setText('Target Angle      :   ' + str(round(Config.Stepper2_Target,3)))
    self.Stepper2_FeedbackValue.setText('Feedback Angle :   ' + str(round(Config.Stepper2_Feedback,3)))

    self.Stepper3_TargetValue.setText('Target Angle      :   ' + str(round(Config.Stepper3_Target,3)))
    self.Stepper3_FeedbackValue.setText('Feedback Angle :   ' + str(round(Config.Stepper3_Feedback,3)))

    self.TargetPosX.setText(str(Config.Target_Pos_x))
    self.TargetPosY.setText(str(Config.Target_Pos_y))
```

Figure 3.46: Code excerpt of the GUI update values.

```
747         # ------------------------------------------------------- CPU TMP
748         if platform.system() == 'Linux':
749             self.output = subprocess.check_output(['sensors'])
750             self.temp1 = self.output.split()[5].decode()
751             self.temp2 = self.output.split()[14].decode()
752             self.CPUTmp1.setText(self.temp1)
753             self.CPUTmp2.setText(self.temp2)
754         else:
755             self.CPUTmp1.setText('only linux')
756             self.CPUTmp2.setText('only linux')
757
758         self.FB_ball = QPoint(400 +(700/400)*Config.cam_x, 400  -(700/400)*Config.cam_y)
759         if self.Target_ball_pos is None:
760             self.Target_ball_pos =  QPoint(400, 400)
761         self.update()
762
763         self.log_string = '\n'.join(Config.Log)
764
765         if self.log_string != self.log_string_old:
766             self.LogBox.setText(self.log_string)
767
768         self.log_string_old = self.log_string
769
770         if not Config.Camera_Runnig:
771             self.CamerashowMask.setEnabled(False)
772             self.CamerashowMask.setChecked(False)
773             self.CamerashowMask.setStyleSheet("background-color: red")
774             Config.Camera_show_Mask = 0
775             self.CamerashowFrame.setEnabled(False)
776             self.CamerashowFrame.setChecked(False)
777             self.CamerashowFrame.setStyleSheet("background-color: red")
778             Config.Camera_show_Frame = 0
779         else:
780             self.CamerashowMask.setEnabled(True)
781             self.CamerashowFrame.setEnabled(True)
```

Figure 3.47: Code excerpt of the save and cancel function.

```
331         timer = QTimer(self)
332         timer.timeout.connect(self.UpdateInformativeValues)
333         timer.start(50)
```

Figure 3.48: Timeout timer for the UpdateInformativeValues function.

To update the *Value label* objects shown in Table 3.4, their text needs to be changed by using *.setText()*. This is done in the function *UpdateInformativeValues* from Figure 3.46 and 3.47. Figure 3.46 does not contain every object, but only a small sample because of repetitiveness. The labels are updated from the values stored in Config.py

The *UpdateInformativeValues* function also updates some of the objects used in the GUI. From line 756 in Figure 3.47 the feedback coordinates for the *FB_ball* are updated from the Config.py file. From line 761 to 765 the LogBox object from Table 3.8 is updated. The object takes a string list from Config.py as an argument for *.setText()*. A problem is that the LogBox object is scrollable but the scroll is reset each time the object is updated. An if sentence is used to update the object when a change is detected in the string list. There is also an if sentence on line 768, to set the value and disable the button used to show the camera frame and mask.

To make the function execute, a trigger has to be created. This trigger object is shown in Figure 3.48. In the trigger object, a time object is created from *QTime*. This object is connected to the *UpdateInformativeValues* function whenever the timer is timed out. The timeout is set to 50ms. This will execute the *UpdateInformativeValues* function every 50ms.

3.6 Main.py

To merge the application, a main Python script is created. This script initialises the GUI, background threads and processes. The structure and data flow of the main operations in the application is shown in Figure 3.49.

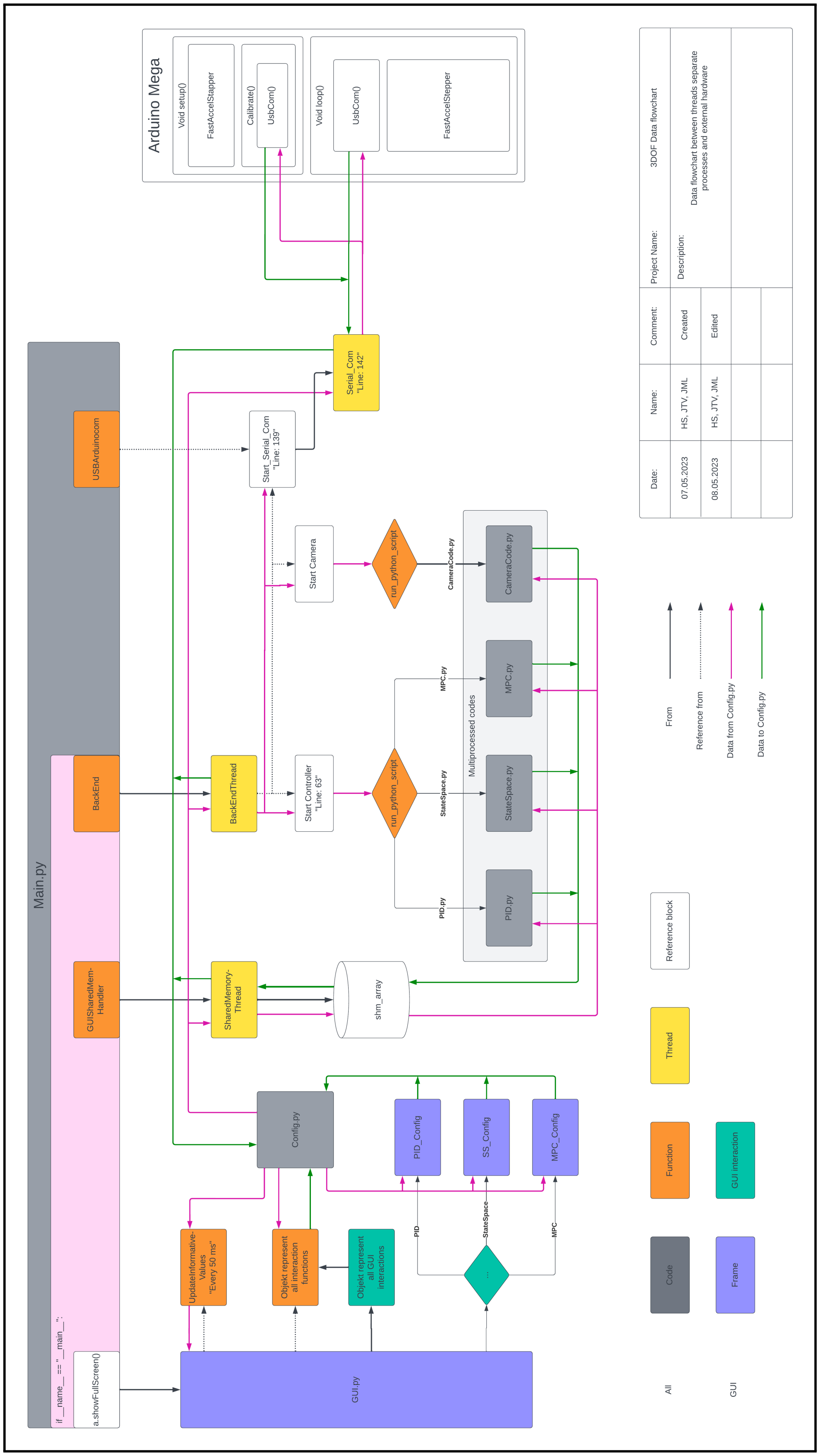


Figure 3.49: Data flowchart and structure.

The Main.py script contains three threaded operations. These are shown in the yellow boxes in Figure 3.49. It is important to thread the background operations when using a GUI. This is to make sure the GUI doesn't freeze under large operations. Further sections describe all operations and declarations created in the Main.py script.

3.6.1 Config.py

Config.py is a script where all the shared variables are declared. This script is used as a module import in other Python scripts to overwrite or read the variable values.

3.6.2 Shared Memory Declaration

The different controller and camera scripts which are used are multi-processed. This is to make sure they can run on separate central processing unit cores with more resources available. This causes them to not share the same memory space as the Main.py script. In order for the scripts to communicate, a shared memory block is created which the different scripts can access.

```
19    lock = multiprocessing.Lock()
20    shm_array = C.Shm_array
21    shm = multiprocessing.shared_memory.SharedMemory(create=True, size=shm_array.nbytes)
22    shm_array = np.ndarray(shm_array.shape, dtype=np.float16, buffer=shm.buf)
23
```

Figure 3.50: Declaration of the shared memory.

The shared memory object is declared as *shm* by using the *multiprocessing.shared_memory .SharedMemory* package on line 21 in Figure 3.50. Next, a NumPy array named *shm_array* is created where the *buffer* argument sets the array buffer to be the shared memory buffer. This means that the NumPy array is backed by the shared memory block of the *shm* object. The *shm_array* is predefined in Config.py as *Shm_array = np.zeros(25, dtype=np.float16)*, this is done so that its size only has to be defined one time. This also makes it possible to edit its size without editing the other scripts. It's important to note that the elements in the *shm_array* need to be float values. There can not be any integer or boolean values stored in the array.

```python
# -------------------------- shm clear
def cleanup_shm():
    shm.close()
    shm.unlink()
```

Figure 3.51: Code excerpt of clearing and destroying the shared memory.

To protect against memory leaks, there needs to be a way to remove the shared memory block. This is done in the function *cleanup_shm* from Figure 3.51.

3.6.3 GUISharedMemHandler

To read and write values in the shared memory block a function is used. This function works as a data handler which transports data in and out of the shared memory block.

```python
def GUISharedMemHandler():

    PrevT = time.time()
    while not C.exit_flag:
        CurrT = time.time()
        dt = CurrT - PrevT
        PrevT = CurrT
        if dt == 0:
            C.SharedMem_Hz = 9999.99
        else:
            C.SharedMem_Hz = round(1 / dt,2)
        #--------------------------------------------------- shm handler
        lock.acquire()
        shm_array[0] = C.Target_Pos_x
        shm_array[1] = C.Target_Pos_y
        shm_array[2] = C.Target_Vel_x
        shm_array[3] = C.Target_Vel_y

        C.cam_x = shm_array[4]
        C.cam_y = shm_array[5]
        C.cam_x_vel = shm_array[6]
        C.cam_y_vel = shm_array[7]

        C.U_rad_x = shm_array[8]
        C.U_rad_y = shm_array[9]

        shm_array[10] = C.PID_P_x
        shm_array[11] = C.PID_D_x
        shm_array[12] = C.PID_I_x

        shm_array[13] = C.SS_k1_x
        shm_array[14] = C.SS_k2_x

        shm_array[15] = C.MPC_Q_xp
        shm_array[16] = C.MPC_Q_xv
        shm_array[17] = C.MPC_R_Vin
        if C.Start:
            C.Controller_Hz = shm_array[18]
        else:
            C.Controller_Hz = 0
        shm_array[19] = C.MPC_SnapShot

        shm_array[20] = C.No_Ball
        shm_array[21] = C.Camera_show_Frame
        shm_array[22] = C.Camera_show_Mask
        shm_array[23] = C.Camera_Pause
        if C.Camera_Runnig:
            C.Cam_Hz = shm_array[24]
        else:
            C.Cam_Hz = 0
        lock.release()
```

Figure 3.52: Code excerpt of the data handler function.

The data handler is shown as the *GUISharedMemHandler* function in Figure 3.52. The specific data handling happens in between the *lock.acquire()* and *lock.release()* methods. These two methods are used to acquire a lock on the *shm_array* object. This is done to make sure that other processes can't modify its content while the current process is running.

3.6.4 BackEnd

To start and stop the different background operations a backend function is created, This function will handle the start and stop of multi-processed and multi-threaded code and also the data recording.

```python
def run_python_script(script, shm):
    # Execute the script and pass the shared memory object as an argument
    shared_data = {'shared_data': shm}
    exec(open(script).read(), shared_data)
```

Figure 3.53: Function to execute en external script.

The function shown in Figure 3.53 is used to execute an external Python script. While executing this script it also passes the shared memory object *shm* from Figure 3.50 into the script.

```python
def BackEnd():
    Running_Flag = True

    PrevT = time.time()

    while Running_Flag:

        CurrT = time.time()
        dt = CurrT - PrevT
        PrevT = CurrT
        if dt == 0:
            C.BackEnd_Hz = 9999.99
        else:
            C.BackEnd_Hz = round(1 / dt,2)
        # ----------------------------------------------------------- Start Controller selected
        if (C.Start) and not (C.Running):

            ControllerCode = C.Control_Mode +'.py'
            if C.Control_Mode == 'No Control Mode selected':
                C.Log.insert(0,'No Controller selected!')
            else:
                C.Log.insert(0,'Starting Controller: ' + ControllerCode)
                p1 = multiprocessing.Process(target=run_python_script, args=(ControllerCode,shm,))
                p1.start()
                C.Running = True

        if not C.Start and C.Running:
            C.Running = False
            C.Log.insert(0,'Stopping Controller')
            p1.terminate()
            C.Stepper1_Target = 0
            C.Stepper2_Target = 0
            C.Stepper3_Target = 0
            lock.acquire()
            shm_array[8] = 0
            shm_array[9] = 0
            lock.release()
```

Figure 3.54: BackEnd function and controller start and stop.

In Figure 3.54 the backend function is defined, further a while loop is created so the code runs continuously. The first operation the *BackEnd* function have is the start and stop of the controller code. It takes the controller code selected from Figure 3.44 and 3.45, and passes that string as a name argument for the controller code to be executed in Figure 3.53.

When the controller code is terminated all the variables it has changed are set back to zero, this includes the values it has written to the shared memory as well.

```
# ------------------------------------------------------------ Start Camera Code
if (C.Camera_Start and not C.Camera_Runnig):
    C.Log.insert(0,'Starting Camera Code')
    p2 = multiprocessing.Process(target=run_python_script, args=("CameraCode.py", shm,))
    p2.start()
    C.Camera_Runnig = True

if C.Camera_Runnig and C.cam_x == 1 and C.cam_y == 9 and C.cam_x_vel == 9 and C.cam_y_vel == 9:
    #C.Camera_Runnig = False
    C.Log.insert(0,'No connection to camera')
    lock.acquire()
    shm_array[4] = 0
    shm_array[5] = 0
    shm_array[6] = 0
    shm_array[7] = 0
    lock.release()

if not C.Camera_Start and C.Camera_Runnig:
    C.Log.insert(0,'Stopping Camera Code')
    p2.terminate()
    C.Camera_Runnig = False

    lock.acquire()
    shm_array[4] = 0
    shm_array[5] = 0
    shm_array[6] = 0
    shm_array[7] = 0
    lock.release()
```

Figure 3.55: Camera start and stop.

The second operation the *BackEnd* function handles is the start and stop of the camera code. It follows the same procedure as the controller start and stop, but it has an included section to check if the camera code has no frame. If the camera code has no frame it will terminate automatically after passing the numbers 1, 9, 9, 9 to the shared memory.

```
# ------------------------------------------------------------ Stop Multithreading code at exit
if C.exit_flag:
    try:
        Running_Flag = False
        p1.terminate()
    except:
        pass
    try:
        p2.terminate()
    except:
        pass
```

Figure 3.56: Code excerpt of multi-processed code termination.

To make sure the parallel processes are terminated upon exit the *BackEnd* function is the last to stop and the *exit_Flag* variable tries to terminate the processes before it sets its own running flag to False. A try operation is used so that no error occurs if it tries to terminate a process that isn't running.

```
if C.Start_Serial_Com and not C.Serial_Com_Running:
    C.Serial_Com_Running = True

    Serial_Com = threading.Thread(target=USBArduinocom)
    Serial_Com.start()

if not C.Start_Serial_Com:
    C.Serial_Com_Running = False

time.sleep(0.01)
```

Figure 3.57: Start and stop the communication thread.

To start the communication with the Arduino card, a thread is started from the *USBArduinocom* function shown in Figure 3.13. This is done in Figure 3.57. Since the code is threaded, it has access to the same memory space as the application. It is then possible to use the *Serial_Com_Running* variable to stop the thread. This variable stops the while loop inside the *USBArduinocom* function, and the thread task will stop automatically after that.

3.6.5 if __name__ == "__main__":

if __name__ == "__main__": is a condition statement used in Python to check whether the current script is being run as the main program. This is commonly used in multiprocessing applications to not spawn unintended processes.

```python
if __name__ == "__main__":
    app = QApplication(sys.argv)
    a = GUI()

    ShareMemoryThread = threading.Thread(target=GUISharedMemHandler)
    ShareMemoryThread.start()

    BackendThread = threading.Thread(target=BackEnd)
    BackendThread.start()

    a.showFullScreen()
    atexit.register(cleanup_shm)

    sys.exit(app.exec_())
```

Figure 3.58: Application execution.

The part of the Main.py script that starts the application is shown in Figure 3.58. Here, an instance of the QApplication class is created. This instance needs to be created to manage the application's control flow and handle events. Next, an instance of the GUI class from the GUI.py script is created.

After a GUI class instance is created, a thread is started for each of the two background operations. These operations are the functions declared in Figure 3.52 and 3.54. These functions have a while loop inside them to make them run repetitively. The while loop has a condition variable that is meant to stop the loops when the application is closed.

To create and show the actual GUI window frame the *.showFullScreen()* is called on the GUI instance. One can also use *.show()* but this has a big offset on the Ubuntu operating system. The *clean_shm* function from Figure 3.51 is set to be executed when the application is exited. This is done by using the *atexit* library.

Finally by calling *sys.exit(app.exec_())* the Qt event loop is started by the argument and will be terminated by the function when the event loop calls for it or an error occurs.

3.7 Data collection and calculation

To save data for analysis two data collection methods have been built into the application. Both methods start when the *Record Data* button is activated. The first method takes the variable values stored in Config.py and stores them continuously until the *Record Data* button is deactivated. The other method is specifically designed for the MPC code and takes a snapshot of the current predicted angles and states. This snapshot also captures the feedback values in the same time window as the MPC prediction horizon.

3.7.1 Data Recorder

The data recorder is located in the *BackEnd* thread from the Main.py script. This recorder records all the target values and feedback values. These values are then stored in a text file stored in the "`/SavedData/Record`" directory for later analysation.

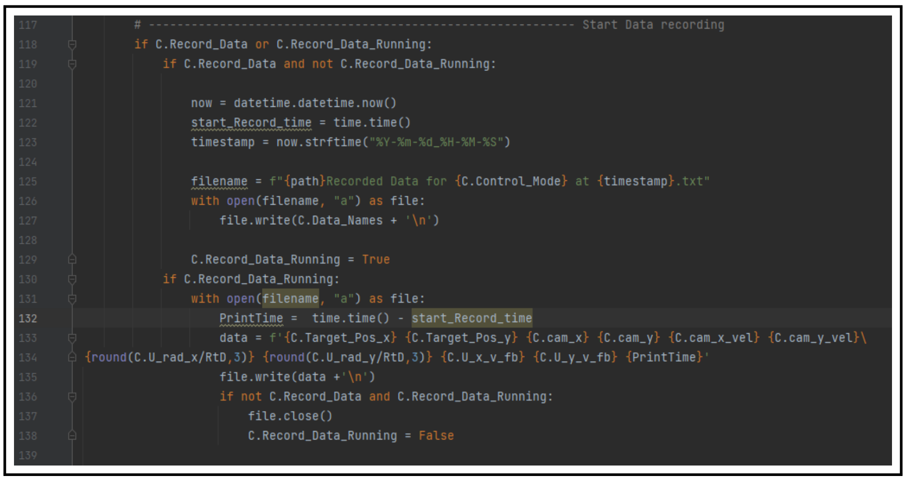


Figure 3.59: Desktop run script.

3.7.2 Snap Recorder

The snap recorder is located in the MPC.py code, this recorder will take a snapshot of the prediction horizon to the MPC one second after the *Record Data* button is pressed. It will also record the new values in and out of the MPC so that it is possible to see if the prediction is true.

```python
# ----------------------------------------------------- Take a snapshot of the prediction
if MPC_SnapShot > 0.0 and not SnapShotRunnig:
    StartTime = time.time()
    FB_U_x = []
    FB_P_x = []
    FB_V_x = []
    FB_U_y = []
    FB_P_y = []
    FB_V_y = []
    TimeList = []
    SnapShotRunnig = True

if SnapShotRunnig:
    RunTime = time.time() - StartTime
    if RunTime > 1:
        if not SnapTaken:
            Predicted_U_x = angle_list[0, :]
            Predicted_S_P_x = state_list[0, :]
            Predicted_S_V_x = state_list[1, :]
            Predicted_U_y = angle_list[1, :]
            Predicted_S_P_y = state_list[2, :]
            Predicted_S_V_y = state_list[3, :]
            SnapTaken = True
        FB_U_x.append(u_traj[0])
        FB_P_x.append(s_error[0])
        FB_V_x.append(s_error[1])

        FB_U_y.append(u_traj[1])
        FB_P_y.append(s_error[2])
        FB_V_y.append(s_error[3])
        TimeList.append(RunTime-1)

    if RunTime > 2:
        SnapShotRunnig = False
        WriteSnap = True
        FileCreated = False
        n_predict = 0
        n_feedback = 0
```

Figure 3.60: Desktop run script.

```python
if WriteSnap:
    if not FileCreated:
        now = datetime.datetime.now()
        start_Record_time = time.time()
        timestamp = now.strftime("%Y-%m-%d_%H-%M-%S")
        filename_Predicted_x = f"{path} MPC SnapShot Predicted x {timestamp}.txt"
        filename_Predicted_y = f"{path} MPC SnapShot Predicted y {timestamp}.txt"
        filename_FB_x = f"{path} MPC SnapShot True x {timestamp}.txt"
        filename_FB_y = f"{path} MPC SnapShot True y {timestamp}.txt"
        FileCreated = True

    if (len(Predicted_U_x) > n_predict):
        with open(filename_Predicted_x, "a") as file_1:
            data1 = f'{Predicted_U_x[n_predict]} {Predicted_S_P_x[n_predict]} {Predicted_S_V_x[n_predict]}'
            file_1.write(data1 + '\n')

        with open(filename_Predicted_y, "a") as file_2:
            data2 = f'{Predicted_U_y[n_predict]} {Predicted_S_P_y[n_predict]} {Predicted_S_V_y[n_predict]}'
            file_2.write(data2 + '\n')

        n_predict += 1


    if (len(FB_U_x) > n_feedback):
        with open(filename_FB_x, "a") as file_3:
            data3 = f'{FB_U_x[n_feedback]} {FB_P_x[n_feedback]} {FB_V_x[n_feedback]} {TimeList[n_feedback]}'
            file_3.write(data3 + '\n')

        with open(filename_FB_y, "a") as file_4:
            data4 = f'{FB_U_y[n_feedback]} {FB_P_y[n_feedback]} {FB_V_y[n_feedback]} {TimeList[n_feedback]}'
            file_4.write(data4 + '\n')
        n_feedback += 1

    else:
        file_1.close()
        file_2.close()
        file_3.close()
        file_4.close()
        WriteSnap = False
```

Figure 3.61: Desktop run script.

The snap recorder code is shown in Figure 3.60 and 3.61. The recorder is written in such a way as to minimize its impact on the loop time of the MPC. A new data point is stored in a list every loop and when the prediction window time has been reached one value from the list will be written to a .txt file every loop. This is done so as not to be stuck in a loop in the writing phase. This is a bit time-consuming and the whole process Will take around 3-4 seconds to complete. The .txt file is stored in the "`/SavedData/Snap`" directory.

3.7.3 Recording Of Miscellaneous Data

There is also additional data that wants to be stored, this is done by replacing the data that is being saved from the data recorder by changing the data variable in Figure 3.59.

The first set of data to be recorded is the refresh rate of the different parallel processes. For this the data variable is set to "*data = f'{C.BackEnd_Hz} {C.SharedMem_Hz} {C.UartCom_Hz} {C.Cam_Hz} {C.Controller_Hz}'*". The second set of data recorded is the latency between the *GUISharedMemHandler* thread, the *MPC.py* process and the Arduino. Additionally, the hertz for these two processes is also added to check for any correlation. This is done by defining a sine wave in the *GUISharedMemHandler* function, passing that through the MPC.py process back to the *GUISharedMemHandler*, and then writing the sine wave to all the stepper motors and recording the feedback angles. For this the record data will be "*data = f'{C.sine} {C.sine_controller} {C.Stepper1_Feedback} {C.Stepper2_Feedback} {C.Stepper3_Feedback} {C.SharedMem_Hz} {C.Controller_Hz} {PrintTime}'*", one also needs to change the data sent in the *USBArduinocom* function shown in Figure 3.13 to "*data = struct.pack('ffff', C.sine_controller, C.sine_controller C.sine_controller, float(C.Calibrate_Arduino))*".

3.8 Khadas Setup

This section will describe how the Khadas vim3 is configured.

3.8.1 Operating System

The Khadas VIM3 originally comes with an Android operating system, this is not sufficient for this use case so an Ubuntu OS is going to be installed. To install this OS the USB flash tool and installation guide provided in the Khadas document[3] is going to be used. The different Ubuntu OS versions supported by this method on the Khadas can be acquired from Khadas download page[4]. From here the *vim3-ubuntu-20.04-gnome-linux-4.9-fenix-1.5-230425-emmc.raw.img.xz* is used.

3.8.2 Boot File

The screen that is being used has a resolution of 1280x800. The Khadas should be able to auto-detect this resolution but that wasn't the case. To get the Khadas set to the correct resolution, the *hdmi_autodetect* in the `/boot/env.txt` file has to be disabled and set to a fixed resolution. This is done by following the steps in the Khadas documents about resolution[5].

[3] https://docs.khadas.com/products/sbc/vim3/install-os/install-os-into-emmc-via-usb-tool

[4] https://dl.khadas.com/products/vim3/firmware/ubuntu/emmc/

[5] https://docs.khadas.com/products/sbc/vim3/configurations/hdmi-resolution#tab__configuration-file

The predefined usage of the general-purpose input/output pins is also located in the `/boot/env.txt` file. the UART setup for the GPIO pins should be turned on by default but this is possible to check by following the Khadas documents for the devise tree overlay[6] and UART[7].

3.8.3 Deployment on Khadas

To transfer the application to the Khadas the Secure Shell connection deploy method in Pycharm was used, this method requires the Pycharm professional edition.

For this to work the Khadas need to be connected to the same network as the computer running Pycharm, this is done with an Ethernet cable connected to the Khadas. Now it is possible to add the Python interpreter used on the Khadas to Pycharm, to do this go to File -> Settings... and locate the section shown in Figure 3.62.

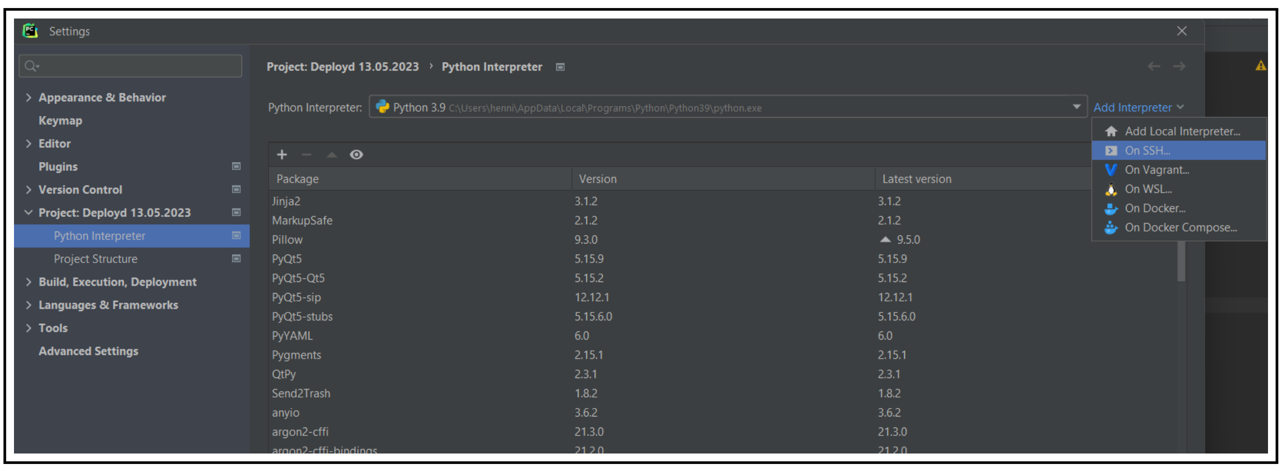


Figure 3.62: Adding the Khadas interpreter.

When in the Python Interpreter section on Pycharm go to "Add Interpreter" and select "On SSH...". From here, a window to set up the SSH connection will pop up, this is shown in Figure 3.63 and 3.64.

[6] https://docs.khadas.com/products/sbc/common/configurations/device-tree-overlay#tab__vim1233ledge1

[7] https://docs.khadas.com/products/sbc/vim3/applications/gpio/uart

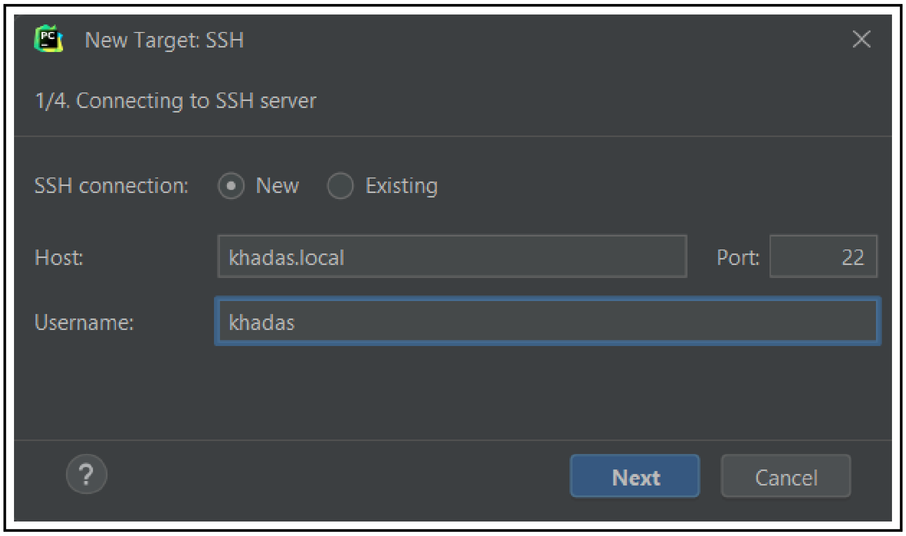


Figure 3.63: SSH connection.

After pressing next in Figure 3.63 a password authentication window shows, the password here is "Khadas".

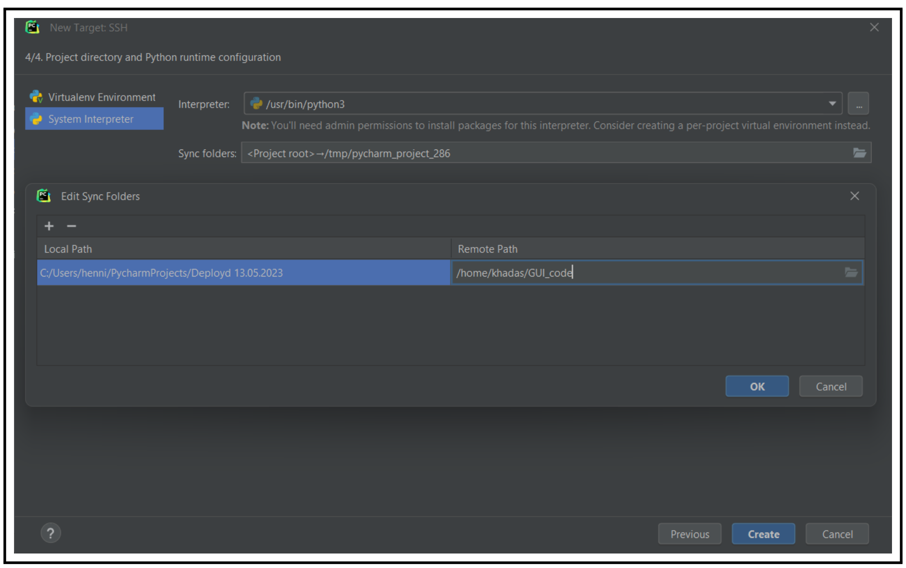


Figure 3.64: Sync path.

The final step is to add the external interpreter's location and the application's external sync path. The interpreter to be used is the Khadas python3 system interpreter. Where everything is located is shown in Figure 3.64. Here the "Remote Path" is the directory where the whole Pycharm project is going to be uploaded to.

By using this method it is possible to deploy and run code on the Khadas from an external machine by using the ordinary run method in Pycharm, Pycharm will print any values or error messages as that if it was running on the local machine. This is helpful when testing different testing codes like the camera code, I2C or UART. This method also has its limitations where running code that opens graphical events on the Khadas does not work. So it is not possible to test the GUI code or show the camera frames from an external run.

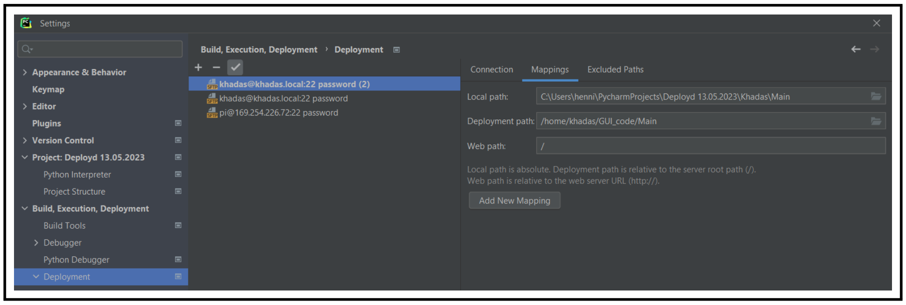


Figure 3.65: Deployment path.

At first, the whole Pycharm project is synchronized to the Khadas. This is done by using the deploy method rather than the sync method. This is to make sure the files don't get downloaded back to Pycharm after a name change or deletion in Pycharm.

The deployment path is changed so that the application code located in the Main folder is deployed to its corresponding Main folder on the Khadas. The changes needed to be done are shown in Figure 3.65 under "Local path:" and "Deployment path:".

For forced deployment or sync go to Tools -> Deployment located in the top right toolbar on Pycharm. The best option here is to upload the open window or the current file. The current file will be the file window in Pycharm that was last clicked on.

3.8.4 Desktop Script

To be able to easily run the application for the motion platform, a desktop script has been created. The desktop script executes the Main.py script with the Python 3.8 interpreter. The desktop script sets the home directory of the Main.py script to be the project directory. A Desktop image is attached to the desktop file font from the project directory icons folder. The run script is displayed in Figure 3.66. To make this script executable, a *chmod +x Controller_GUI.desktop* command needs to be called in the Ubuntu terminal. It is important to not use sudo privileges when doing this, if so the scripts can only be executed by a sudo user.

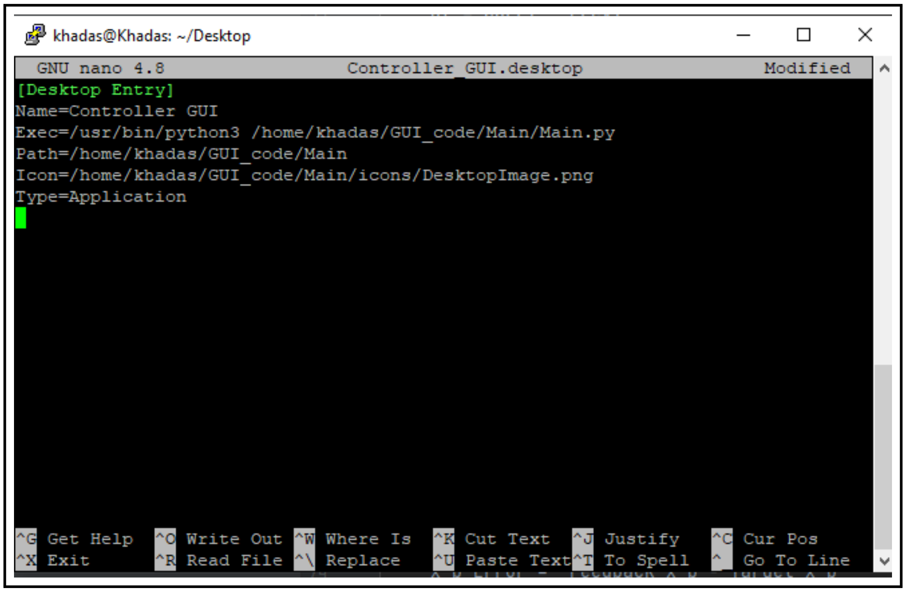


Figure 3.66: Desktop run script.

3.8.5 Shared Folder

The folder that holds the data recorded is made available as a shared folder, which makes it possible to access the folder by connecting the Khadas to a Windows computer using the same WIFI or by using an Ethernet cable. Then by opening *File explorer* on your Windows computer and heading to the *Network* folder located in the bottom left, it is possible to open the folder by typing "`\\khadas.local`" in the directory path. This will direct you to the guest folder shown in Figure 3.67.

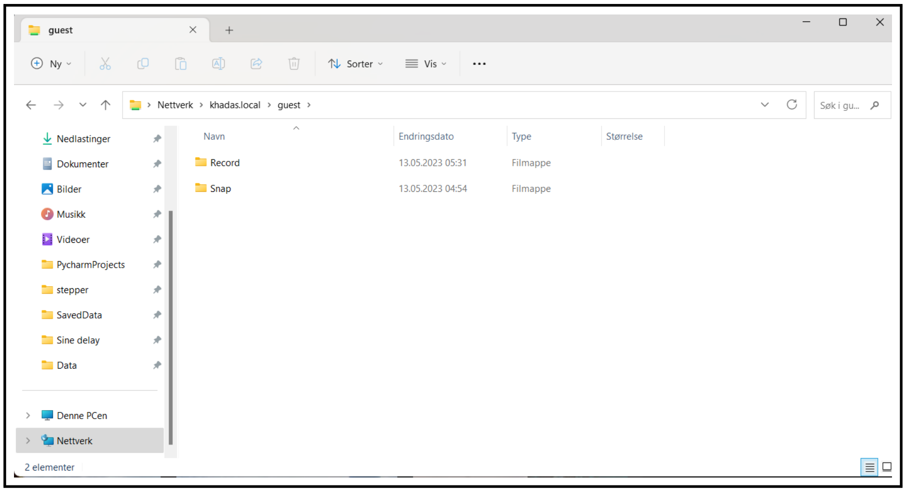


Figure 3.67: Shared folder on Windows.

To make this possible a program named *samba* is installed, this program allows Linux-based operating systems to share files with Windows computers. To install and configure the Samba program follow these steps.

1. Install samba by typing the following in the terminal "`sudo apt-get install samba`"
2. Open the samba configure file by typing "`sudo nano /etc/samba/smb.conf`" in the terminal.
3. Add the lines shown in Figure 3.68 to the end of the config file.
4. Save and close the file.
5. Restart the samba service by typing "`sudo service smbd restart`" in the terminal.

```
[guest]
        path = /home/khadas/GUI_code/Main/SavedData
        read only = no
        guest = yes
        guest only = yes
```

Figure 3.68: Samba code.

Chapter 4

Results

This chapter presents the results of the design, GUI, and control algorithms. A video demonstration can be found here[1].

4.1 Design

This section presents the final render of the platform in the design process.

Figure 4.1: Final render of the assembled platform.

[1] https://youtu.be/brYy_x_78rQ

4.2 GUI Results

This section presents the final GUI frame and config frame for the controller. It also contains a description of the functionality behind the buttons in the GUI.

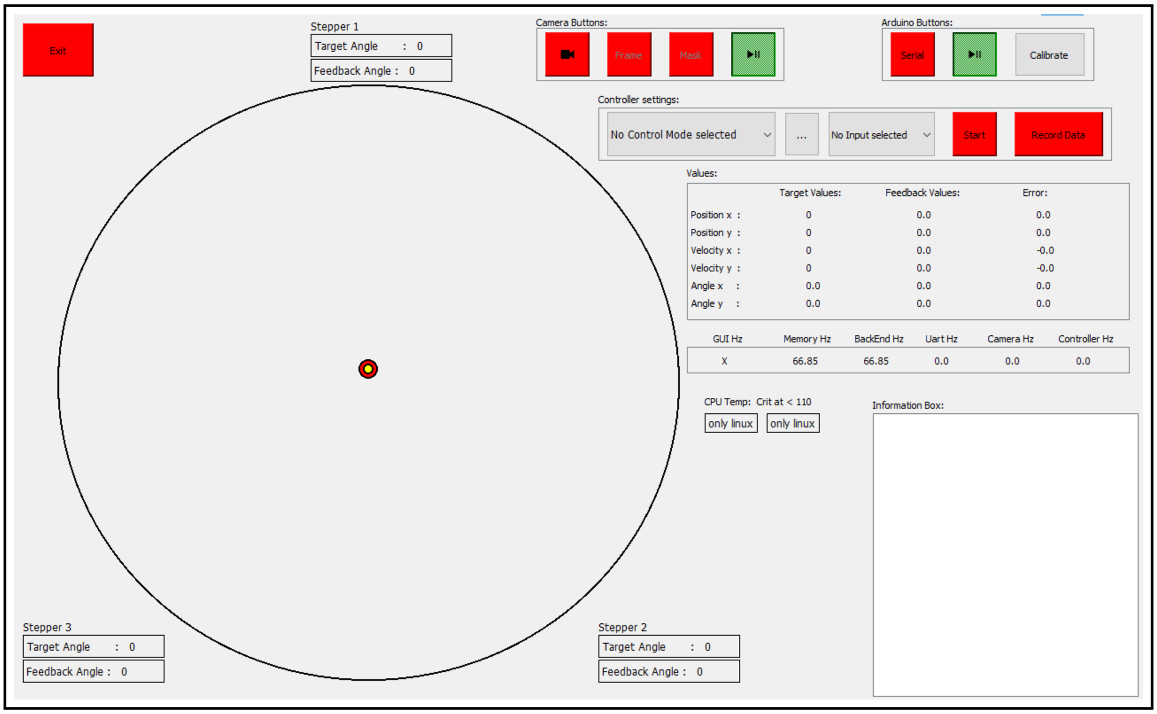


Figure 4.2: Figure of the complete GUI.

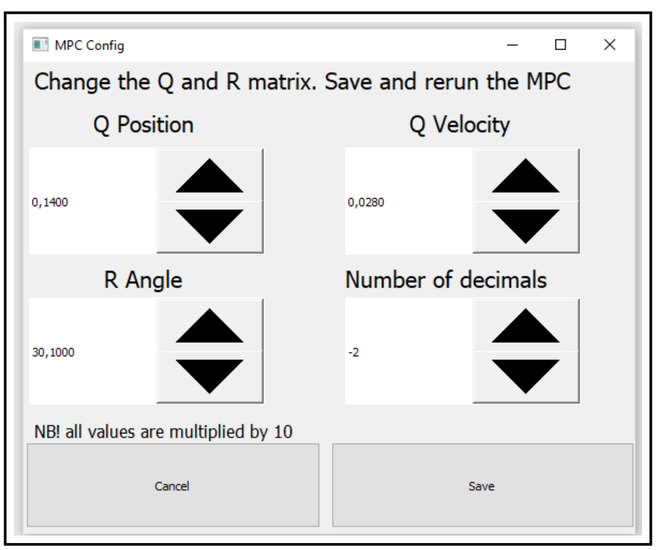


Figure 4.3: Figure of the resulting config frame for the MPC.

Field	Description
Q Position	Position weight in Q matrix for both X and Y axis
Q Velocity	Velocity weight in Q matrix for both X and Y axis
R Angle	Angle weight in R matrix for both X and Y axis
Number of decimals	Decimal position. Decide which decimal position changes apply to

Table 4.1: Table describing the functions of MPC config window.

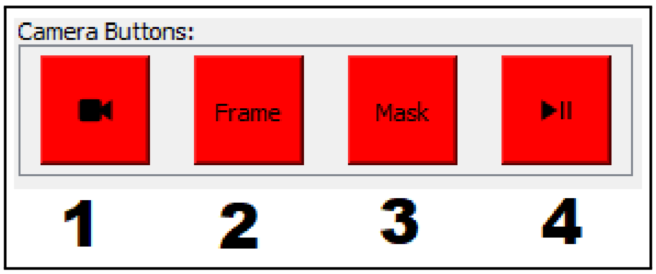


Figure 4.4: Figure of camera buttons.

Button	Description
Button 1: Initialize	Initialize and start the camera code
Button 2: Frame	Show the normal camera feed
Button 3: Mask	Show the HSV camera feed
Button 4: Pause	Pause or unpause the camera values

Table 4.2: Table describing the functions of each camera button.

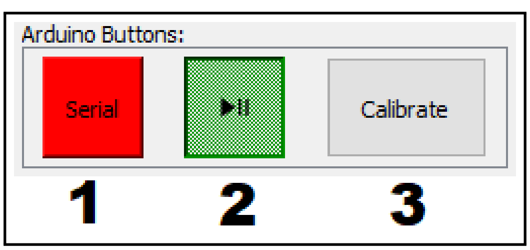


Figure 4.5: Figure of Arduino buttons.

Button	Description
Button 1: Serial	Start the serial communication between Khadas and Arduino
Button 2: Pause	Pause or unpause the serial communication
Button 3: Calibrate	Calibrate the internal stepper motor values

Table 4.3: Table describing the functions of the Arduino buttons.

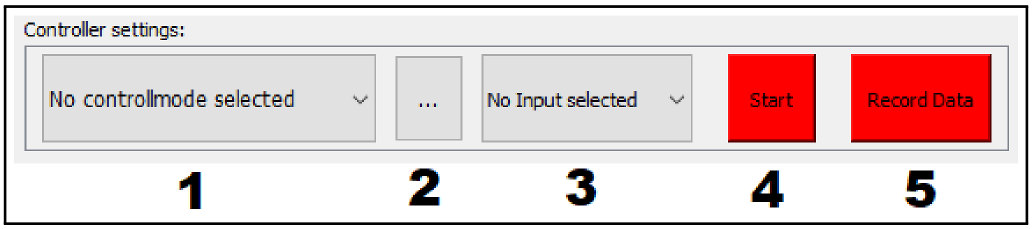


Figure 4.6: Figure of Arduino buttons.

Button	Description
Button 1: Controller	Choose which controller to use. The start button must be off to change controller
Button 2: Settings	Pause or unpause the serial communication
Button 3: Input	No function
Button 4: Start	Start the controller. Must be off to change controller
Button 5: Record Data	Start data collection of target values and feedback values

Table 4.4: Table describing the functions of the controller buttons.

4.3 MPC Results

This section presents plots of the results gained by balancing a ball on the motion platform. All results are recorded with the same tuning parameters. Half of the results are gathered while running the application on the Khadas. The other half of the results are gathered while running the application on a computer. The first two plots in the MPC results present the actual position of the ball compared with the target position. The last two plots of the MPC results present and compare the feedback angles of the X-axis with the target angle of the X-axis.

4.3.1 Khadas VIM3 MPC Results

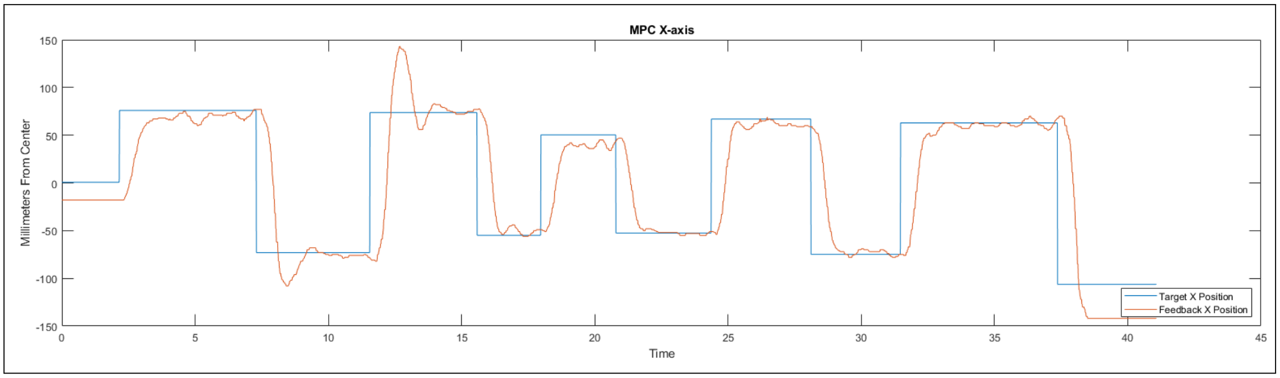


Figure 4.7: A plot of the target and feedback position of the ball on the X-axis on the Khadas VIM3.

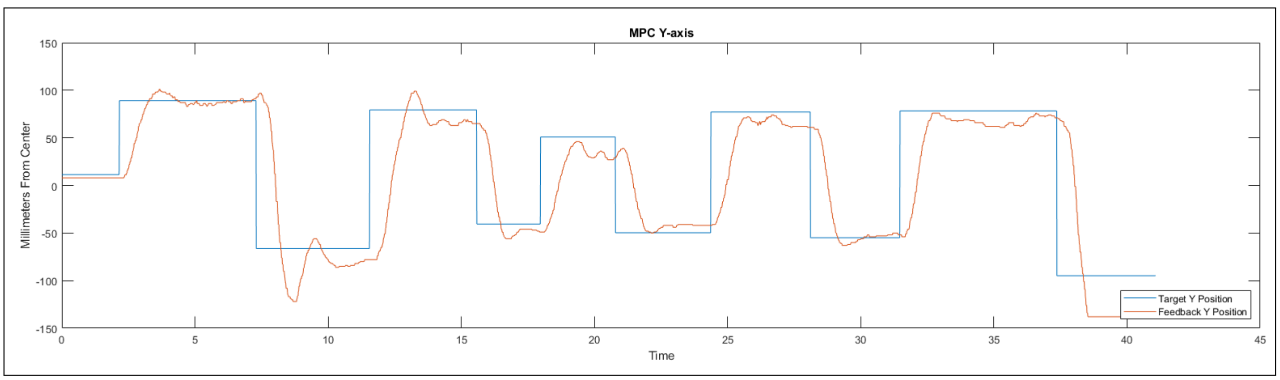


Figure 4.8: A plot of the target and feedback position of the ball on the Y-axis on the Khadas VIM3.

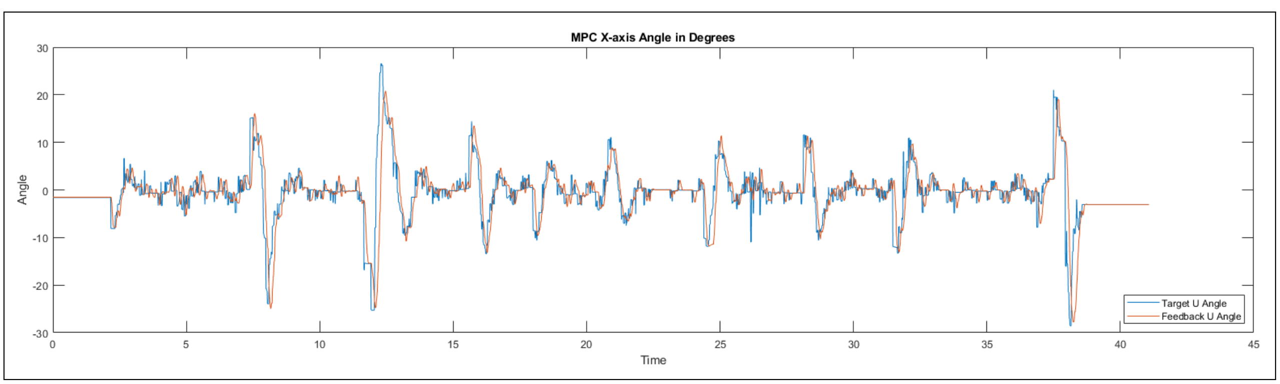


Figure 4.9: A plot of the target and feedback angle of the X-axis on the Khadas VIM3.

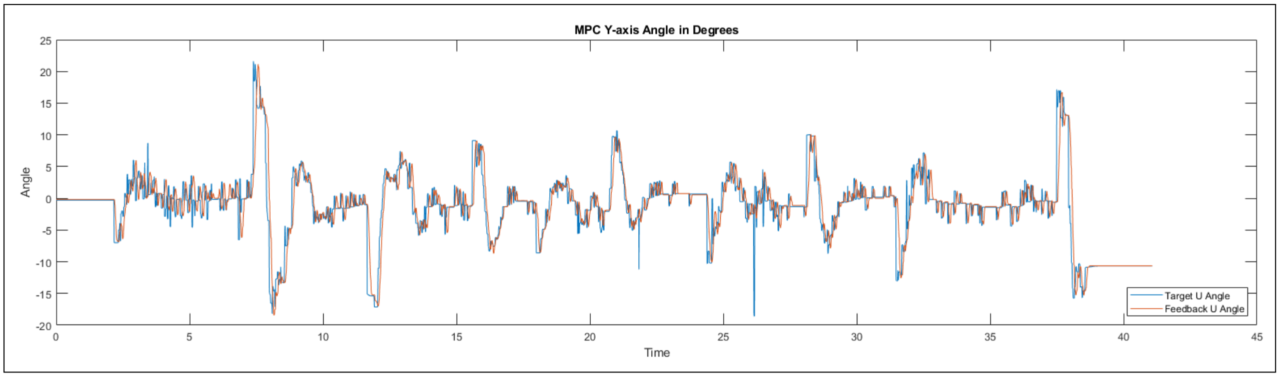


Figure 4.10: A plot of the target and feedback angle of the Y-axis on the Khadas VIM3.

4.3.2 Computer MPC Results

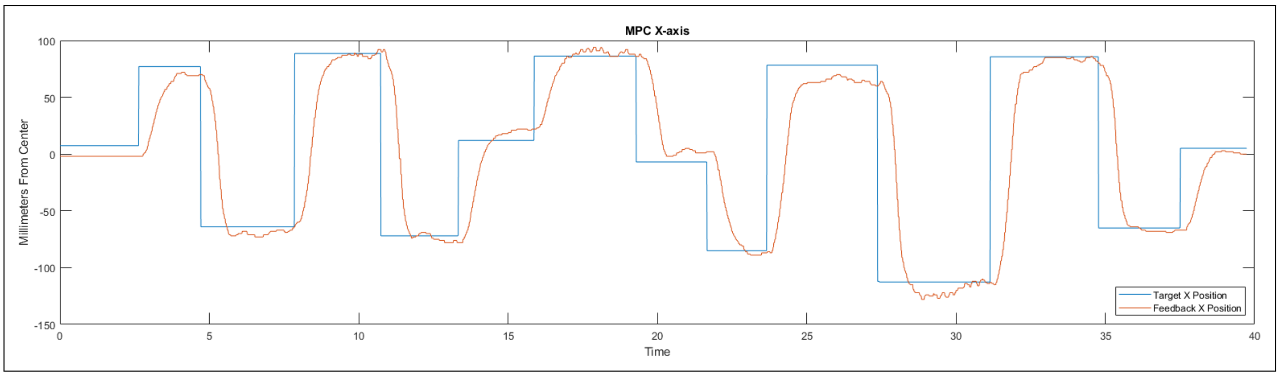


Figure 4.11: A plot of the target and feedback position of the ball on the X-axis on the Computer.

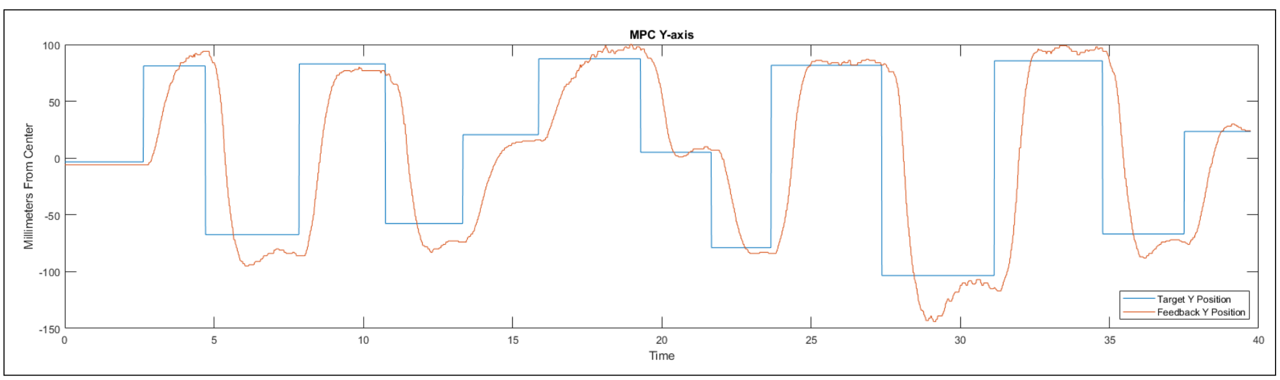


Figure 4.12: A plot of the target and feedback position of the ball on the Y-axis on the Computer.

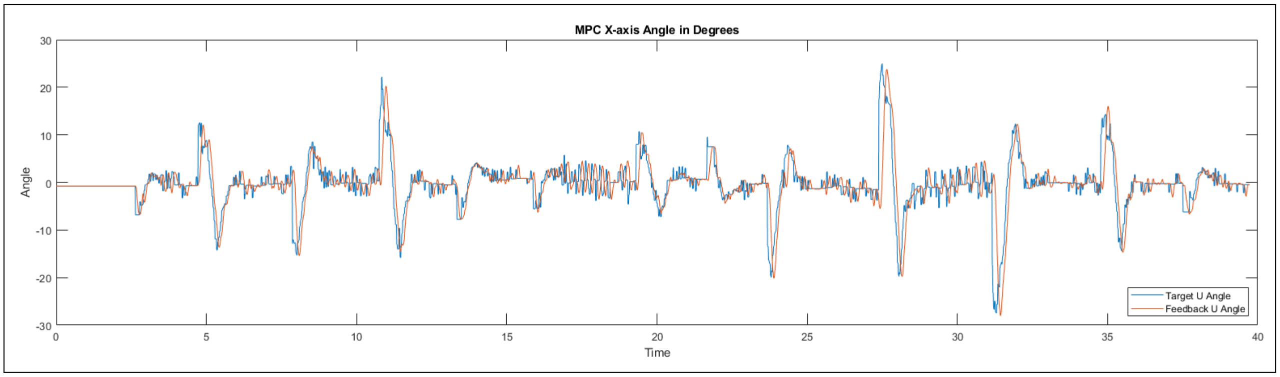


Figure 4.13: A plot of the target and feedback angle of the X-axis on the Computer.

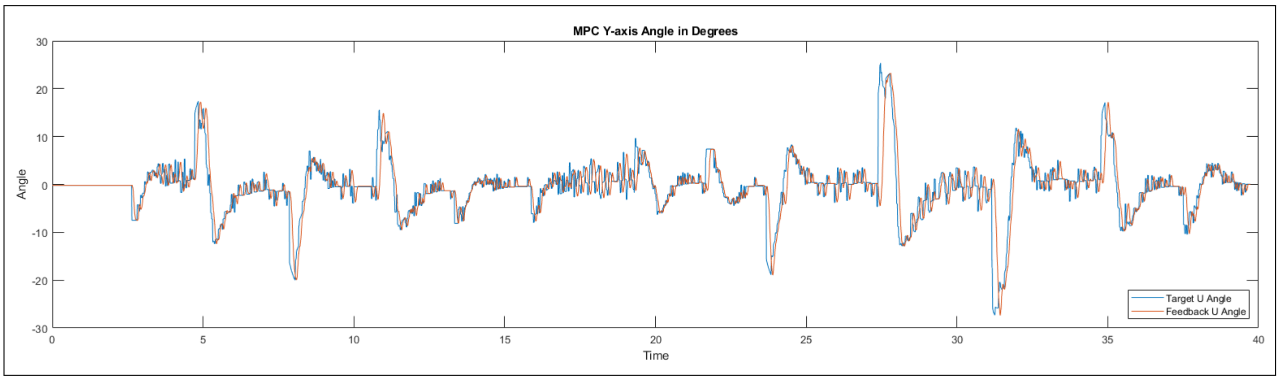
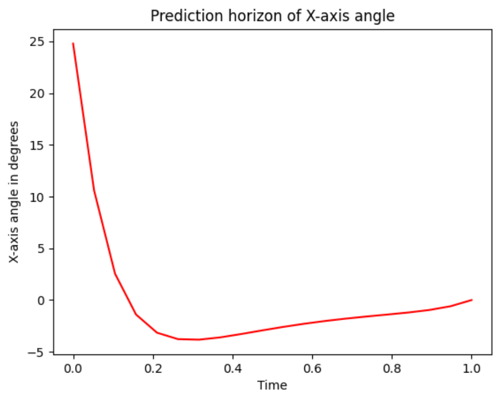


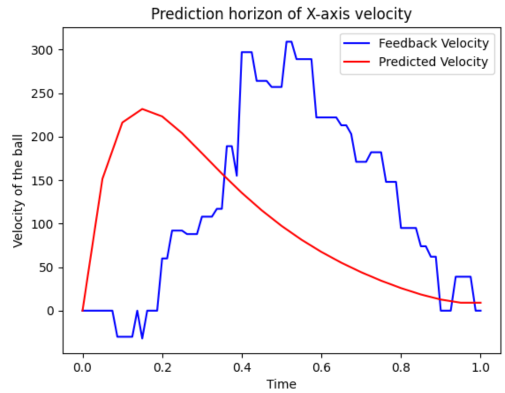
Figure 4.14: A plot of the target and feedback angle of the Y-axis on the Computer.

4.3.3 MPC Prediction Horizon

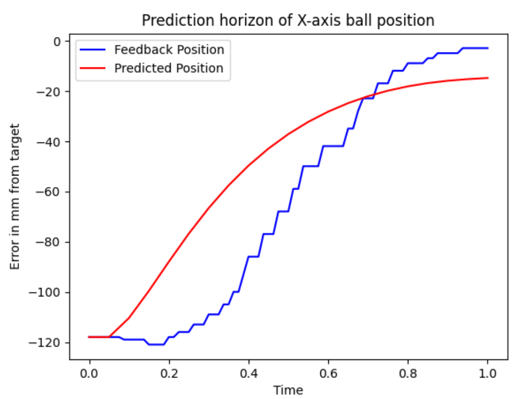
These results present the prediction horizons the MPC calculates with an error in the position of -120. The position and velocity graph also shows the corresponding feedback in position and velocity for comparison.



(a) MPC example angle prediction horizon.



(b) MPC example velocity prediction horizon and feedback velocity.



(c) MPC example position prediction horizon and feedback position

4.3.4 Latency Stress Test

This section presents the results from the MPC stress test where the ball position has a fast and big change in position. These results show the refresh rate of the memory handler, controller, and a sine wave. The first plot presents the latency in the controller. The second plot presents the corresponding target and feedback to illustrate the difference in feedback at low latency.

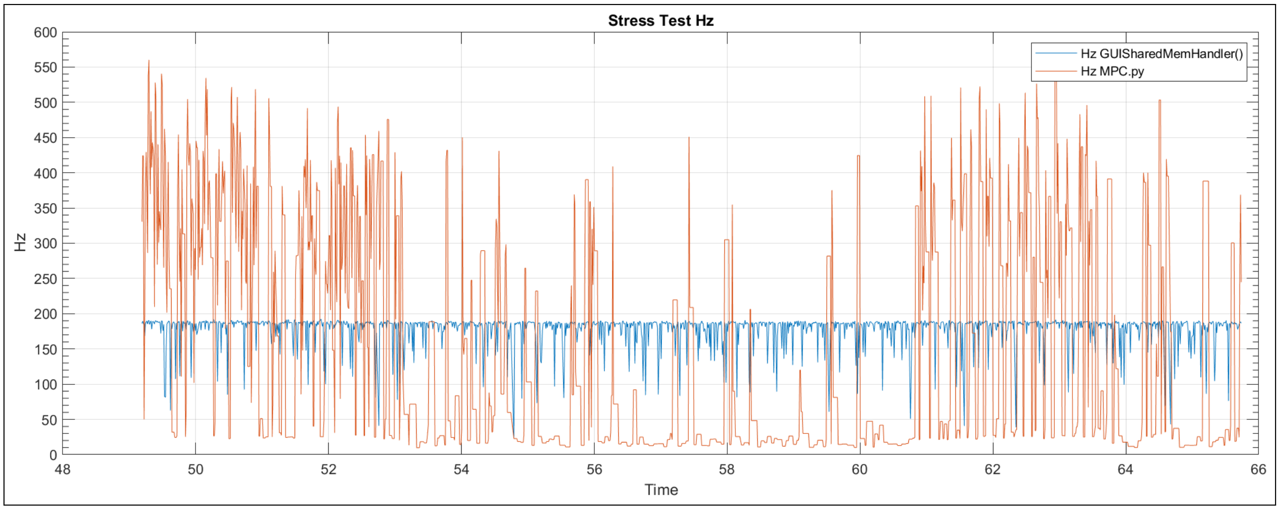


Figure 4.16: Process refresh rate from the stress test.

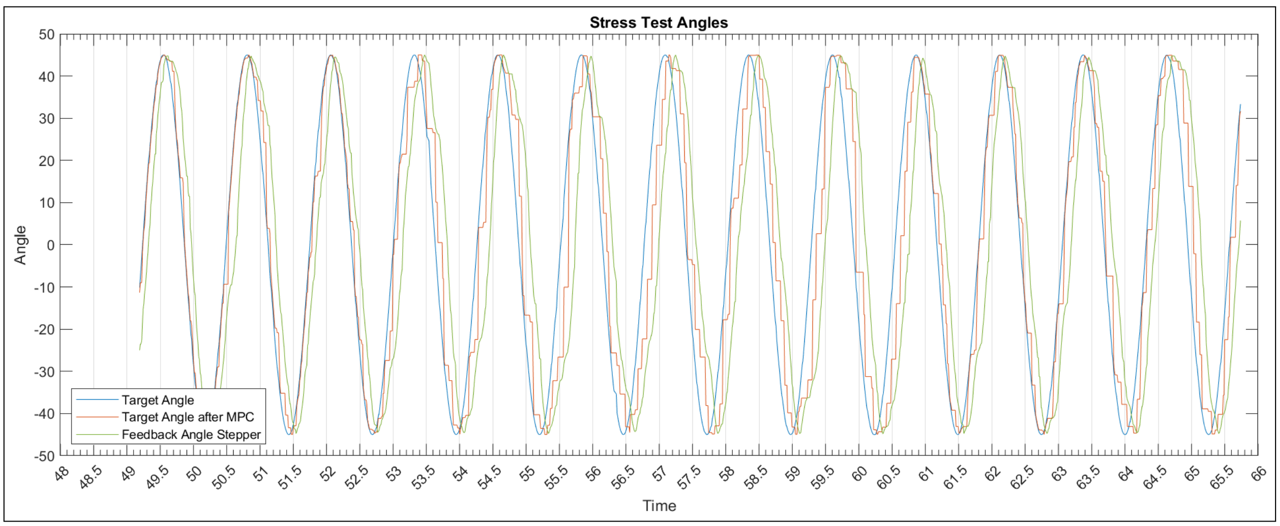


Figure 4.17: A plot of the target and feedback angle from the stress test.

Chapter 5

Discussion

This chapter will discuss the design results, the code, the MPC results and future improvements. It highlights both the solutions and the challenges that were encountered during this project.

5.1 Design Results

The final platform was fairly close to the original concept. During testing, the dimensions were slightly altered, but this did not significantly change the design. The few modifications made were necessary due to practicality and mechanical stability issues.

5.1.1 Part Selection & Changes

Stepper Motors

We selected stepper motors instead of servo motors for this project because of the cost factor. Servo motors with the same specs and capabilities as stepper motors can be more than 3 times as expensive. Because of this, stepper motors were seen as a better option than servo motors for managing the budget. The Dual Shaft Nema 23 (23HS30-2804D) from Figure 3.8 which was used, suited our project well. The increased amount of steps made the rotation smoother, but the acceleration slower. The finished project runs on 1600 micro-steps, to ensure a balance between smoothness and precision. This was determined through trial and error. Running below 1600 steps caused vibrations because each step was too large to operate smoothly. Running above

1600 steps made the platform slow, because each step is small, which affects the acceleration.

The stepper motors are so-called open loop stepper motors since they do not feedback a value to the motor drives. A closed-loop stepper motor compares the number of output steps to how many steps were completed and compensates based on the error. Since the original plan was to use encoders as a feedback source, there was no need to use closed-loop stepper motors. This is why open-loop stepper motors are used in this project. In the end, no encoders were used because of the noise and instability issues discussed in the encoder issue sections 5.1.3 and 5.1.3. This made the only solution to use the open-loop stepper motors. However, since the motors were accurate, no problems occurred with using open-loop motors and steps were rarely missed.

Camera

This section discussed the results and problems that occurred during the selection of the camera. Initially, the Khadas 8MP HDR camera was chosen to be used as the camera for object detection. This camera seemed to be the best option at first since it was the most compatible with the Khadas VIM3 single-board computer. When the camera was tested it was observed that the FOV listed in the specification was diagonal, and not horizontal and vertical. This meant that the camera did not have a large enough FOV to observe the entire platform. To solve this a new lens seemed to be the best option. When the wider lens arrived, it was observed that the Khadas camera used non-standardized threads for its interchangeable lens mount. This meant a new lens could not be attached. In the end, the Arducam camera was the only option left. This camera proved to be a much better fit and had multiple lens options to choose from.

5.1.2 Mechanical Improvements

Modifications were required because the motor legs slipped and flexed. A thicker leg was first modelled and printed, but this had little effect. Experimentation on the infill percentage when 3D printing was also attempted unsuccessfully. Another option was to shorten the leg. This would reduce the torque force on the leg and therefore reduce the flexing. The original length was 10 cm and was reduced to 7.5 cm, which would lead to a 25% torque reduction. To increase its strength even further, the leg was also made thicker.

The platform top's circumference was decreased when shortening each of the three legs. The reduction caused the top's diameter to decrease from 45cm to 40cm. This meant that the area was reduced from 1590 cm^2 to 1257 cm^2. As a result, the plate had less room to flex because there was less space between the fastening joints. The thickness of the plates was also increased from 4 mm to 6 mm, which significantly improved stability and eliminated flexing of the plate.

5.1.3 Electrical Improvements

Encoder Issues and Arduino implementation

Precise and accurate control is essential when operating a 3DOF platform. This is why encoders were used to feedback the axial rotation of the stepper motors. The encoders were wired directly to the Khadas board, to begin with. At first, the readings were noisy and fluctuated in value. Earthing wires were applied to the stepper motors to eliminate the noise 3.6. After some testing, another issue occurred, since the Khadas I/O ports could not handle the interrupts from the encoders. An Arduino (Mega 2560 Rev3) 3.2 board was added to work as a dedicated encoder signal reader. The Arduino forwards the encoder feedback to the Khadas. This solution gave steady and reliable feedback.

In retrospect, this was a positive change since it meant all code on the Khadas could be programmed in Python. This meant that only one extra script had to be made on the Arduino to handle the stepper motor control. The only disadvantage to this solution is the extra layer of serial communication between the Khadas and the Arduino.

Noise Issues

Having steady and dependable signals between the electronics is one of the most important aspects to guarantee precise control. A lot of electrical noise was present during the project, which made it difficult to maintain reliable control. It was challenging to identify the precise components that caused the noise. The noise effect was caused by the stepper motors. This occurs because the stepper motors are using large coils (stators) which emit high magnetic forces to rotate the axial shaft. These magnetic forces will affect electronics nearby by causing EMI (Electromagnetic Interference). This effect was influencing both the encoder and limit switch readings. The encoder had issues with counting rotations even if the motor was not rotating. By grounding all components properly, this issue was reduced but still present.

5.1.4 Planning and Ordering parts

Key components of the original design were ordered early in the project's time frame. This was done to ensure having all the parts on time. To ensure quick delivery of the parts, it became crucial to use geographically convenient logistical vendors. This was successful, and the building phase got underway as scheduled. Later on, however, some components did not function as intended. For example, the camera field of view was not great enough in the vertical direction. As a result, a new camera was required and therefore ordered. Little progress was accomplished prior to the new camera's arrival because the project depended on it.

Another example was the special shielded encoder cables. These were ordered to eliminate the noise on the encoders by grounding the protective shield on the cable. The delivery time on these was 3 weeks, which meant that the 12-pin cables could not be finished, and testing had to continue with noise issues.

5.2 The Code

This section discusses the results of the source code and the choice of programming language.

5.2.1 Programming Results

The code was designed to be flexible and capable of running on various computers and operating systems with minimal to no modifications. It was also designed to be modular where operations can be modified or changed without compromising other operations. The code is also scaleable to a point where new features and operations can be added with minimal modifications. By using Python, one can easily achieve all of these goals without issues occurring with OS-specific compilers and builds.

5.2.2 Main Codebase Language

The initial plan was to use C++ to get the most out of the iteration speed for the software to the motion platform. This was because C++ is seen as a better option for high-performance computations, and speed. While researching and testing, Python gave surprisingly fast computational results. Python also had the best availability and easiest implementation of packages. With reasonably fast computational speeds, easy implementation, and better package options for MPC and camera implementation, Python was chosen as the main language for the implementation of algorithms.

By choosing Python as the algorithm language, the time consumption of programming and implementing a working system became substantially lower. This gave more time to fine-tune and adjust design, in order to make the motion platform as user-friendly as possible for others to use and modify. The motion platform is also seen as a tool for students to implement, and test their own controllers, and by using Python as a programming language, further implementations of code are substantially easier.

5.2.3 GUI

The GUI provides a good user experience with options for people to observe how each part of the system functions. The GUI is made with user-friendliness in mind, such that the GUI is as self-explanatory as possible. The left side of the GUI provides an additional control experience which allows others to change the target position of the ball. This allows users to experiment with the ball in a more controlled environment, rather than applying disturbance to the ball. The right side of the GUI provides the user with controller, camera, recording, and communication options. This frame also provides a wide variety of feedback values, which allows users to analyze the system in more detail. Further details and explanations of the GUI can be seen in section 4.2.

After the application was installed on the Khadas, it was discovered that the GUI alone consumed a significant amount of resources. Due to a lack of CPU resources, the application crashed when the camera and MPC code were launched. Additionally, the camera and MPC code would still be running as a result of this action. Investigation revealed that the GUI utilized between 80 and 85 % of all CPU cores. This was discovered to be the result of the functions *.self.update()* and *.UpdateInformativeValues* being called every 50 ms and at each paint event. A significant reduction in CPU usage was observed by altering the *.self.update()* function. These alterations consisted of moving it to its own update function with a 100ms timeout and also removing it from the paint event functions and *.UpdateInformativeValues* function. The *.UpdateInformativeValues* function refresh time was also increased from 50ms to 550ms. These changes resulted in the GUI using between 50 and 60 % of the CPU, which allowed stable operation at full capacity. It's important to keep in mind that changing these variables only affects how frequently graphics are refreshed.

5.2.4 Ball Tracking Algorithm

The ball tracking algorithm works as intended. By observing the video feed while running the platform, one can see that the ball is easily detected, and a circle is painted around it. To arrive at the exact HSV bounds for the colour detection to only track the ball, it is possible to display the video feed in HSV colour space. By adjusting the bounds while displaying the HSV feed, the most suited bounds were discovered. These were found to be within $([0,88,85])$ as the lower bound, and $([13,255,201])$ as the upper bound. This boundary for colours lets the algorithm detect the red ball, without any noise, or unwanted background colours being detected.

There are only two possible errors that can occur while tracking. One of the issues is that the platform is placed directly under a sharp light source. This light source has a high probability to cause unwanted errors in the tracking, depending on the strength of the lighting. The other source for errors is if similar objects appear in the frame. For example, a hand or face from a human. Although human skin isn't red, it is still detected because of the orange colour tone.

5.3 MPC Results

This section discusses the results and graphs gained from the MPC algorithm while trying to balance the ball. This includes which parameters were used to gain the results and comparisons between the Khadas VIM and a computer. The results are gathered with a sampling time of 0.05 seconds and a horizon length of 20 steps. By using this horizon, updates to the prediction happen every 50 milliseconds for a total of one second. The weighting matrix Q was tuned with $([1.7, 0.55, 1.7, 0.55])$ across the diagonal and the weighting matrix R was tuned with $([305, 305])$ across the diagonal. To get the most accurate input for the angle, the second value from the returned angle array is sent to the motors. These angles compensate for the ball's predicted position 50 ms ahead of time from when the measurement is taken and eliminates some of the error between measurement and actual response. These tuning values seemed to give stable results and were gathered by testing different parameters through trial and error. The results are gathered by using a red ping-pong ball with minimal mass.

The overall iteration speed for the MPC algorithm is surprisingly fast. The iteration speed averages around 1.5 ms on the computer and 5 ms on the Khadas VIM. This speed is better than expected and manages to keep up with the measurement frequencies. These results indicate that the rule sets and concepts of DCP and DPP included in the CVXPY packages work as expected. One of the downsides to running the MPC on the Khadas is its unstable calculation speed. Even though the Khadas averages 5 ms, it tends to vary greatly in its speed, which can cause the control over the ball to lag. The calculation speeds range from 2 ms up to about 15 ms. This is still in an acceptable range for the algorithm to keep up with the measurement frequencies but may cause the motion platform to lose control over the ball.

5.3.1 Ball Position

The results and graphs of the MPC algorithm performance for the ball position are shown in section 4.3 in Figure 4.7, 4.8, 4.11, and 4.12.

The graphs for the Khadas VIM3 results show varying responses. The ball seems to reach the target position at an average of 1 second and settles between 1.5 to 2 seconds. This is relatively fast considering the distance the ball has to travel to go from one end of the platform to the other. A small concern in the result for the Khadas is that it tends to be a bit slow and overshoot the target. The reason for this could be internal lag when recording high speeds with the Khadas, bad tuning, or errors in the velocity estimate.

The graphs for the computer results show promising performance of the MPC. The ball manages to reach the target at about 1 second similar to the Khadas and settles at about 1.5 seconds. The computer graphs also show stable positioning of the ball, with minimal corrections.

When comparing the graphs of the Khadas and computer results, it is possible to see that the computer performs a little better. The computer has little to no tendencies to overshoot the target and stabilizes the ball faster than the Khadas. This is to be expected since the computer has a significant advantage in computational power compared to the Khadas. With this in mind, the Khadas still gives surprisingly good results considering it is a single-board computer and its small size.

When comparing the data sets used to plot these graphs, a small difference in the frequency of readings was noticed. The Computer records values from the algorithms such as feedback position, time, and target at around 91 Hz, while the Khadas seem to record at a frequency of 83 Hz. This difference in frequency is not major but signifies a general slowness in the Khadas system.

5.3.2 Angles

The results containing the graphs for target and feedback angle are shown in section 4.3 in Figure 4.9, 4.10, 4.13, and 4.14. These graphs present the calculated angle the MPC algorithm finds optimal, and the actual feedback angle on the platform. By observing the graphs it is possible to infer that the actual angle on the platform follows the target angle closely. When looking at the stability of the angles, it is possible to observe oscillations when the angles should stabilize the ball. This is a repercussion of unstable measurements or bad tuning. When the MPC algorithm is tuned to perfection these oscillations should be near non-existent, indicating the ball has reached its target and is idle. To achieve better stability, better estimations of position and speed through filtering should be implemented, and further tuning should be experimented with.

When comparing the Khadas VIM results with the computer results, both show equal results. This indicates there is no additional delay in the Khadas from when the angle is calculated until the platform reaches its target position. The platform uses 100 ms to achieve its target position from when the angle is calculated on both the Khadas and the computer.

5.3.3 Prediction Horizon

In section 4.3.3 graphs containing the angle, velocity, and position for a prediction horizon are shown. The MPC calculates a unique prediction horizon based on each measurement of the states as initial conditions. The graphs only show the prediction horizon for the X-axis since the Y-axis has the same results. These prediction horizons are calculated with states containing -120 in positional error, and 0 in velocity error. The predicted horizon for the position also contains the feedback of the ball with -120 in positional error. On observation, the MPC gives an accurate representation of the prediction horizons. The MPC initially calculates and predicts the angle as aggressive before giving a small response in the opposite direction to brake the velocity and stabilize the ball. The velocity and position prediction results give corresponding actions based on the angle the MPC calculates. It is also possible to observe the actual feedback of the ball follows the predicted trajectory closely, with only small deviations. This is a good sign that indicates the system models are accurate.

5.3.4 Stress Test Results

During the MPC testing, a latency in control feedback was sometimes observed. Upon closer inspection, it was noted the ball had to be held in the same location for a few seconds before the platform reduced its latency. This was found to be caused by the MPC algorithm using hot-start. This means that the MPC starts searching for an optimal solution from the previously measured position. If there is a big difference in position, the MPC will have a larger range to search through before it finds the optimal solution. This results in the MPC using a long time to calculate the optimal control response. No significant latency was observed under normal operation.

A stress test of the MPC algorithm was performed to analyze the latency of the loop time. This was accomplished by using large, quick ball movements and sending a sine wave through the MPC algorithm before sending it to the Arduino code, which is then used to record the latency. The resulting frequency and sine wave can be seen in Figure 4.16 and 4.17. By comparing Figure 4.16 and 4.17 it is possible to observe the MPC refresh rate drop under large, rapid ball movements and how that affects the sine wave. Upon closer inspection, a latency of 100 ms is observable under normal operation. The latency peaks at approximately 200ms when a large positional error occurs.

5.4 Future Improvements & Developments

This section will discuss how the project can be further developed and which features that can be improved.

5.4.1 Camera Distortion & Plane Coordinates

The motion platform is currently using an uncalibrated fisheye camera to gather the position of the ball. The disadvantage of using such a camera is the distortion outside of its centre. While this distortion is not significant enough to cause problems while the platform is level, it does cause significant errors in the positional measurements while the platform is at a large angle. This is due to the ball changing height, and since the fisheye camera captures images in an hemispherical area errors in the measurements appear. There are multiple solutions to this problem. One solution is to calibrate the camera, and another is to implement another camera measuring the vertical Z-axis.

There was an attempt to calibrate the camera, but this attempt was unsuccessful. Fisheye cameras are notoriously hard to calibrate due to the number of models used to obtain such a wide angle of view. The calibration was attempted by using a Python script to capture images of a checkerboard and try to calibrate based on the alignment of squares. This only made the distortion worse and often ended up with a significantly less accurate image representation of the area. Further attempts could be made to improve the distortion with more expertise on the subject.

Another solution is to implement a secondary camera to measure the ball's position along the vertical Z-axis. This solution could be attempted alongside the implementation of control with heave on the Z-axis. By using a mathematical model, the ball's position along the level XY-plane could be offset by its position along the vertical Z-plane.

5.4.2 Kalman Filter

One possibility for future improvement is to implement a Kalman filter. A Kalman filter would be a good substitute for the current method of estimating the velocity by deriving position. The reason for choosing a Kalman filter is due to its ability to account for uncertainties, and estimate current velocity based on position measurements and previous velocity estimates. The Kalman filter was not attempted during this project due to time constraints but could prove to be a good substitute and improvement over current methods. Other state estimators could also be implemented.

5.4.3 Tuning

Based on the current results that are gathered, further tuning and experiments on the MPC can still be done to improve the results. The MPC has a vast amount of possible configurations in tuning parameters that can be experimented with, and tuning the algorithm to perfection could take an endless amount of time. As it currently stands using a Khadas single-board computer limits the number of variables that can be used in the prediction horizon. If one were to use something with a greater amount of computational power, one could extend the horizon length and increase the number of time steps. The computational requirements are the only constraining factor that limits some of the tuning capabilities. One could also experiment with algorithms that tune the MPC parameters at a faster rate than a human. An example of this could be a genetic algorithm.

Chapter 6

Conclusions

In conclusion, the final platform closely resembled the original concept with only slight modifications required to improve practicality and mechanical stability. The camera selection required some experimentation and ultimately the Arducam camera proved to be the best fit. The legs of the platform were shortened and made thicker to reduce torque and flexing issues, while the top plate was made thicker to improve stability. Encoder and noise issues were also encountered and addressed with the addition of an Arduino board for encoder feedback, and earthing wires to reduce electrical noise from the stepper motors. Overall, the final platform design met the requirements for precision control and stability necessary for a 3DOF platform. The motion platform also fulfils the goal of being user-friendly and modifiable for other students.

In addition to fulfilling the design criteria, the controller results also proved to be good. The results obtained from the Khadas VIM3 and computer graphs show that the MPC algorithm can control the motion platform successfully. Although the computer outperforms the Khadas in terms of stability and overshoot, the Khadas gives good results considering its small size and lower computational power. In terms of angle stability, both the Khadas and the computer show equal results, indicating no additional delay in the Khadas. However, there are some oscillations observed when the angles should stabilize the ball, indicating small oscillations in the measurements or bad tuning. The prediction horizon results show that the MPC algorithm accurately predicts the ball's trajectory based on the initial conditions. By this one can infer that the MPC is implemented successfully and works as intended.

6.1 Further Work

To improve the project several options are available. These improvements are listed below:

- Implement a Kalman filter for velocity estimation.
- Add an additional camera for measurements and control along the vertical Z-axis.
- Tune the MPC further to improve stability and settling time.

Bibliography

[1] Summary of ship movement. https://web.archive.org/web/20111125015923/http://www.pomorci.com/Zanimljivosti/Ship%27s%20movements%20at%20sea.pdf. (Accessed on 04/24/2023).

[2] Stepper online datasheets. https://www.omc-stepperonline.com/dual-shaft-nema-23-bipolar-1-9nm-269oz-in-2-8a-3-2v-57x57x76mm-4-wires-23hs30-2804d. (Accessed on 05/15/2023).

[3] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[4] The CVXPY Authors. Welcome to cvxpy 1.3 — cvxpy 1.3 documentation. https://www.cvxpy.orgl, Present. (Accessed on 04/01/2023).

[5] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[6] Morten Hovd. A brief introduction to model predictive control, Marc 2004. (Accessed on 04/01/2023).

[7] k.ivoutin. Hsv color space cylinder. http://flickr.com/photos/ivoutin/1501340505/sizes/o/in/set-72157602264055525/, 2007.

[8] Matt. 3dof ball on plate using closed loop stepper motors : 13 steps (with pictures) - instructables. https://www.instructables.com/3DOF-Ball-on-Plate-Using-Closed-Loop-Stepper-Motor/. (Accessed on 04/26/2023).

[9] Webjørn Rekdalsbakken. Design and application of a motion platform in three degrees of freedom. 2005.

[10] Webjørn Rekdalsbakken. Kinematics in 3d space, 2021.

[11] Jeffery A. Schroeder. Helicopter flight simulation motion platform requirements. 1998.

[12] Dale E. Seborg, Duncan A. Mellichamp, and Thomas F. Edgar. *Process Dynamics and Control*, chapter 20. Wylie Series in Chemical Engineering. John Wiley & Sons, third edition, 2011. ISBN 9780470646106. URL http://www.worldcat.org/isbn/9780470646106.

[13] tutorialspoint. Shared memory. https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_shared_memory.htm#. (Accessed on 05/11/2023).

[14] Patrick Sebastian, Yap Vooi Voon, and Richard Comley. Colour space effect on tracking in video surveillance. https://d1wqtxts1xzle7.cloudfront.net/87370921/docs-16879897224d22a8ee11166-libre.pdf?1655002471=&response-content-disposition=inline%3B+filename%3DColour_Space_Effect_on_Tracking_in_Video.pdf&Expires=1683561315&Signature=YYgyKlIR1KyQyV~qydoST2wqwcq3CwF0tWQvpCNP3j3m2buVNDz6D2W2pwm6V1VVBnJeT0WO rlb7OpoTNAQ7B0Y64n8WDUDWb7S3duRJJ2HrfxiaRDblcfGm59IBjPNIEq8L3AdzPMu~QXJj d0OXgO1fipi2SVapaaUw3s6~GlDwG2bTmpE7YkiKDdT4K-tyyQ2MzjLDUec2LRJRDQkhreBCYaWAwm3b9Jzj3ALJuPdMQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA, 2010. (Accessed on 05/08/2023).

[15] Jørgen Meland Lund , Henning Sønderland , Jesper Vos. Ais2102 dynamical systems project report - 3dof motion platform, 2022.

[16] Maximilian Schaller, Steven Diamond, Akshay Agrawal, Alan Yang. Cvxpygen. https://github.com/cvxgrp/cvxpygen, Present.

[17] Krzysztof Zarzycki and Maciej Ławryńczuk. Fast real-time model predictive control for a

ball-on-plate process. *Sensors*, 21(12), 2021. ISSN 1424-8220. doi: 10.3390/s21123959. URL https://www.mdpi.com/1424-8220/21/12/3959.

Appendix A

Appendix Folder List

List detailing the contents of the zip folder delivered as an attachment.

- STL files for 3D-printer
  - BearingColumn, column to glue between the bearing and PlatformLeg.
  - CameraBracket, camera bracket inside the enclosure.
  - MotorBracket, bracket to fasten the stepper motors.
  - PlatformBracket, bracket to fasten the PlatformLeg to the platform.
  - PlatformLeg, leg between the StepperLeg and platform.
  - StepperLeg, leg to fasten to the stepper motor.
  - SupportLegBack, support frame at the back of the enclosure.
  - SupportLegLeft, support frame at the front left side of the enclosure.
  - SupportLegRight, support frame at the front right side of the enclosure.
  - SupportTriangleLeft, left side support for the screen.
  - SupportTriangleRight, right side support for the screen.
- DXF files for laser cutter
  - Platform, plexiglass platform.
  - DisplayFrameBottom, bottom support frame for the screen.

  - DisplayFrameOuter, outer frame for the screen.
  - DisplayFrameTop, top support frame for the screen.
  - EnclosureBack, back wall for the enclosure.
  - EnclosureBottom, floor for the enclosure.
  - EnclosureSide, side walls for the enclosure.
  - EnclosureTop, roof for the enclosure.
  - EnclosureTriangle, triangle wall for the front.
  - InnerWallLong, long inside wall for the fan.
  - InnerWallShort, short inside wall for the fan.

- Datasheets, folder containing all datasheets and diagrams for the parts which are used in this project.
- Source code
  - Main
    * CameraCode.py, camera and ball tracking code.
    * Config.py, contains all variables shared between codes.
    * ControllerTemplate.py, template for others to make their own controllers.
    * GUI.py, GUI code.
    * Main.py, main application code.
    * MPC.py, MPC controller.
    * PID.py, PID controller.
    * StateSpace.py, State Space controller.
    * Info.txt, short information regarding the use of the code.
    * MPC_generator, folder containing the Jupyter notebook code for generating MPC C code.
    * icons, folder containing images for GUI.
    * MPC_code, folder containing generated C code for MPC on Windows.

    - MPC_code_Linux, folder containing generated C code for MPC on Linux.

  - MotorControllerV4

    - MotorControllerV4.ino, Arduino code for the motor control.

- Presentations, folder containing the midterm presentation and poster.

- Videos, folder containing the video demonstration and presentation.

Appendix B

Source Code

This chapter includes all the source code required to run the application.

B.1 Main.py

```
#! -usr/bin/

from PyQt5.QtWidgets import QApplication
import sys
import numpy as np
from GUI import GUI
import Config as C
import multiprocessing
from multiprocessing import shared_memory
import threading
import subprocess
import time
import serial
import struct
import datetime
import platform
import atexit

lock = multiprocessing.Lock()
```

```
shm_array = C.Shm_array
shm = multiprocessing.shared_memory.SharedMemory(create=True, size=shm_array.nbytes)
shm_array = np.ndarray(shm_array.shape, dtype=np.float16, buffer=shm.buf)

start_Record_time = time.time()

RtD = (np.pi/180)

path = 'SavedData/Record/'
filename = ''
#----------------------------------------------------

def run_python_script(script, shm):
    # Execute the script and pass the shared memory object as an argument
    shared_data = {'shared_data': shm}
    exec(open(script).read(), shared_data)

def run_cpp_program(program, shared_data):
    # Execute the program and pass the shared memory object as an argument
    subprocess.run([program, str(shared_data[0]), str(shared_data[1]), str(shared_data
    [2])])

def BackEnd():
    Running_Flag = True

    PrevT = time.time()

    while Running_Flag:

        CurrT = time.time()
        dt = CurrT - PrevT
        PrevT = CurrT
        if dt == 0:
            C.BackEnd_Hz = 9999.99
        else:
            C.BackEnd_Hz = round(1 / dt,2)
```

```
    # ------------------------------------------------------------ Start Controller selected
    if (C.Start) and not (C.Running):

        ControllerCode = C.Control_Mode +'.py'
        if C.Control_Mode == 'No Control Mode selected':
            C.Log.insert(0,'No Controller selected!')
        else:
            C.Log.insert(0,'Starting Controller: ' + ControllerCode)
            p1 = multiprocessing.Process(target=run_python_script, args=(ControllerCode,shm,))
            p1.start()
            C.Running = True

    if not C.Start and C.Running:
        C.Running = False
        C.Log.insert(0,'Stopping Controller')
        p1.terminate()
        C.Stepper1_Target = 0
        C.Stepper2_Target = 0
        C.Stepper3_Target = 0
        lock.acquire()
        shm_array[8] = 0
        shm_array[9] = 0
        lock.release()
    # ------------------------------------------------------------ Start Camera Code
    if (C.Camera_Start and not C.Camera_Runnig):
        C.Log.insert(0,'Starting Camera Code')
        p2 = multiprocessing.Process(target=run_python_script, args=("CameraCode.py", shm,))
        p2.start()
        C.Camera_Runnig = True

    if C.Camera_Runnig and C.cam_x == 1 and C.cam_y == 9 and C.cam_x_vel == 9 and C.cam_y_vel == 9:
        #C.Camera_Runnig = False
```

```
        C.Log.insert(0,'No connection to camera')
        lock.acquire()
        shm_array[4] = 0
        shm_array[5] = 0
        shm_array[6] = 0
        shm_array[7] = 0
        lock.release()

    if not C.Camera_Start and C.Camera_Runnig:
        C.Log.insert(0,'Stopping Camera Code')
        p2.terminate()
        C.Camera_Runnig = False

        lock.acquire()
        shm_array[4] = 0
        shm_array[5] = 0
        shm_array[6] = 0
        shm_array[7] = 0
        lock.release()
    # ------------------------------------------------------------ Stop Multithreading code at exit
    if C.exit_flag:
        try:
            Running_Flag = False
            p1.terminate()
        except:
            pass
        try:
            p2.terminate()
        except:
            pass
    # ------------------------------------------------------------ Start Data recording
    if C.Record_Data or C.Record_Data_Running:
        if C.Record_Data and not C.Record_Data_Running:

```

```
                now = datetime.datetime.now()
                start_Record_time = time.time()
                timestamp = now.strftime("%Y-%m-%d_%H-%M-%S")

                filename = f"{path}Recorded Data for {C.Control_Mode} at {timestamp}.txt"
                #filename = f"{path}Recorded Data for sine delay at {timestamp}.txt"
                with open(filename, "a") as file:
                    file.write(C.Data_Names + '\n')
                    #file.write(C.Hz_Names + '\n')
                    #file.write(C.Sine_Name + '\n')

                C.Record_Data_Running = True
            if C.Record_Data_Running:
                with open(filename, "a") as file:
                    PrintTime =  time.time() - start_Record_time
                    data = f'{C.Target_Pos_x} {C.Target_Pos_y} {C.cam_x} {C.cam_y} {C.cam_x_vel} {C.cam_y_vel}\
{round(C.U_rad_x/RtD,3)} {round(C.U_rad_y/RtD,3)} {C.U_x_v_fb} {C.U_y_v_fb} {PrintTime}'
                    #data = f'{C.BackEnd_Hz} {C.SharedMem_Hz} {C.UartCom_Hz} {C.Cam_Hz} {C.Controller_Hz}'
                    #data = f'{C.sine} {C.sine_controller} {C.Stepper1_Feedback} {C.Stepper2_Feedback}\
# {C.Stepper3_Feedback} {C.SharedMem_Hz} {C.Controller_Hz} {PrintTime}'
                    file.write(data +'\n')
                    if not C.Record_Data and C.Record_Data_Running:
                        file.close()
                        C.Record_Data_Running = False

        if C.Start_Serial_Com and not C.Serial_Com_Running:
            C.Serial_Com_Running = True

            Serial_Com = threading.Thread(target=USBArduinocom)
            Serial_Com.start()

        if not C.Start_Serial_Com:
            C.Serial_Com_Running = False
```

```

            time.sleep(0.01)



sine_controller = 0
def GUISharedMemHandler():
    Last_Target_x = 0
    Last_Target_y = 0
    PrevT = time.time()
    while not C.exit_flag:
        CurrT = time.time()
        dt = CurrT - PrevT
        PrevT = CurrT
        if dt == 0:
            C.SharedMem_Hz = 9999.99
        else:
            C.SharedMem_Hz = round(1 / dt,2)
        #C.sine = 45*np.sin(5*time.time())
        #---------------------------------------------- shm handler
        lock.acquire()
        shm_array[0] = C.Target_Pos_x
        shm_array[1] = C.Target_Pos_y
        shm_array[2] = C.Target_Vel_x
        shm_array[3] = C.Target_Vel_y
        #shm_array[2] = C.sine
        #C.sine_controller = shm_array[3]

        C.cam_x = shm_array[4]
        C.cam_y = shm_array[5]
        C.cam_x_vel = shm_array[6]
        C.cam_y_vel = shm_array[7]

        C.U_rad_x = shm_array[8]
        C.U_rad_y = shm_array[9]

```

```
    shm_array[10] = C.PID_P_x
    shm_array[11] = C.PID_D_x
    shm_array[12] = C.PID_I_x

    shm_array[13] = C.SS_k1_x
    shm_array[14] = C.SS_k2_x

    shm_array[15] = C.MPC_Q_xp
    shm_array[16] = C.MPC_Q_xv
    shm_array[17] = C.MPC_R_Vin
    if C.Start:
        C.Controller_Hz = shm_array[18]
    else:
        C.Controller_Hz = 0
    shm_array[19] = C.MPC_SnapShot

    shm_array[20] = C.No_Ball
    shm_array[21] = C.Camera_show_Frame
    shm_array[22] = C.Camera_show_Mask
    shm_array[23] = C.Camera_Pause
    if C.Camera_Runnig:
        C.Cam_Hz = shm_array[24]
    else:
        C.Cam_Hz = 0
    lock.release()

    #-----------------------------------------------------------

    # l = 426.5mm - 123.12, 213,25, (sq(3)*l)/6, l/2
    # l2 = 346.41mm, 99.99, 173.205

    if C.Running:
        #
        t1 = 100*np.sin(C.U_rad_y)*np.cos(C.U_rad_x)
        t2 = 173.2*np.sin(C.U_rad_x)

```

```
            z1 = -t1 - t2
            z2 = -t1 + t2
            z3 =  t1

            z1 = LimitAngle(z1)
            z2 = LimitAngle(z2)
            z3 = LimitAngle(z3)

            C.Stepper1_Target = np.arcsin(z3/75)/RtD
            C.Stepper2_Target = np.arcsin(z2/75)/RtD
            C.Stepper3_Target = np.arcsin(z1/75)/RtD

        sin_y_cos_x = (75 * np.sin(C.Stepper1_Feedback * RtD)) / 100
        sin_x = ((75 * np.sin(C.Stepper2_Feedback * RtD)) + 100 * sin_y_cos_x) / 173.2
        C.U_x_v_fb = round(np.arcsin(sin_x) / RtD, 2)

        c = sin_y_cos_x/np.cos(np.arcsin(sin_x))
        if c > 1:
            c = 1
        elif c < -1:
            c = -1
        C.U_y_v_fb = round((np.arcsin(c))/RtD, 2)
        time.sleep(0.005)



def LimitAngle(Value):
    # 62', +50
    if Value > 70:
        Value = 70
    elif Value < -60:
        Value = -60
    return Value
```

```python

def USBArduinocom():

    PrevT = time.time()
    if platform.system() == 'Windows':
        Com = 'COM3'
    else:
        Com = '/dev/ttyS3'
    try:
        Arduino = serial.Serial(Com, 115200, timeout=1)
    except:
        C.Log.insert(0,'Failed Connection to Arduino')
        return
        pass

    while not C.exit_flag and C.Serial_Com_Running:

        CurrT = time.time()
        dt = CurrT - PrevT
        PrevT = CurrT
        if dt == 0:
            C.UartCom_Hz = 9999.99
        else:
            C.UartCom_Hz = 1 / dt
        # print(dt)


        if C.Paus_New_Arduino_Values:
            data = struct.pack('ffff', 0.0, 0.0, 0.0, float(C.Calibrate_Arduino))
        else:
            data = struct.pack('ffff', C.Stepper1_Target, C.Stepper2_Target, C.Stepper3_Target, float(C.Calibrate_Arduino))
            #data = struct.pack('ffff', C.sine_controller, C.sine_controller, C.sine_controller, float(C.Calibrate_Arduino))

        Arduino.write(data)
```

```
            response = Arduino.readline().decode().split()
            try:
                C.Stepper1_Feedback = float(response[0])
                C.Stepper2_Feedback = float(response[1])
                C.Stepper3_Feedback = float(response[2])
            except:
                pass
    C.UartCom_Hz = 0

# ----------------------------- shm clear
def cleanup_shm():
    shm.close()
    shm.unlink()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    a = GUI()

    ShareMemoryThread = threading.Thread(target=GUISharedMemHandler)
    ShareMemoryThread.start()

    BackendThread = threading.Thread(target=BackEnd)
    BackendThread.start()

    a.showFullScreen()
    #a.show()
    atexit.register(cleanup_shm)

    sys.exit(app.exec_())
```

B.2 GUI.py

```

# 07.05.2023 3 DOF gui code
# Verson 1.2
```

```
#

import subprocess

from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QComboBox, QPushButton, QFrame, QDoubleSpinBox, \
    QSpinBox, QTextEdit, QMainWindow, QMessageBox
from PyQt5.QtGui import QPainter, QPen, QColor, QFont, QIcon
from PyQt5.QtCore import Qt, QPoint, QRect, QTimer
import Config

import platform
RtD = (3.14/180)

class PID_Config(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(700,350,570,440)
        self.setWindowTitle('PID Config')
        # -----------------------------------------------------------------
        self.font1 = QFont()
        self.font1.setPointSize(16)

        self.font2 = QFont()
        self.font2.setPointSize(12)
        # ---------------------------------------------------------- info
        self.Info = QLabel(self)
        self.Info.move(10,5)
        self.Info.setText('Change P, D and I Gains. Save and rerun the PID')
        self.Info.setFont(self.font1)

        self.info2  = QLabel(self)
        self.info2.move(10,335)
        self.info2.setText('NB! all values are multiplied by 10')
        self.info2.setFont(self.font2)
        # ---------------------------------------------------------- Set P gain
```

```
        self.LableP = QLabel(self)
        self.LableP.move(65,45)
        self.LableP.setText('P Gain')
        self.LableP.setFont(self.font1)

        self.PGain = QDoubleSpinBox(self)
        self.PGain.setObjectName(u"doubleSpinBox")
        self.PGain.setGeometry(QRect(5, 80, 220, 100))
        self.PGain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                                 "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.PGain.setDecimals(5)
        self.PGain.setMinimum(0.00000)
        self.PGain.setSingleStep(0.01000)
        # ------------------------------------------------------- Set D gain
        self.LableD = QLabel(self)
        self.LableD.move(360,45)
        self.LableD.setText('D Gain')
        self.LableD.setFont(self.font1)

        self.DGain = QDoubleSpinBox(self)
        self.DGain.setObjectName(u"doubleSpinBox")
        self.DGain.setGeometry(QRect(300, 80, 220, 100))
        self.DGain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                                 "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.DGain.setDecimals(5)
        self.DGain.setMinimum(0.00000)
        self.DGain.setSingleStep(0.01000)
        # ------------------------------------------------------- Set I gain
        self.LableI = QLabel(self)
        self.LableI.move(75, 190)
        self.LableI.setText('I Gain')
        self.LableI.setFont(self.font1)

        self.IGain = QDoubleSpinBox(self)
```

```
        self.IGain.setObjectName(u"doubleSpinBox")
        self.IGain.setGeometry(QRect(5, 220, 220, 100))
        self.IGain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                            "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.IGain.setDecimals(5)
        self.IGain.setMinimum(0.00000)
        self.IGain.setSingleStep(0.01000)

        #------------------------------------------------------ Set decimal
        self.LableDe = QLabel(self)
        self.LableDe.move(300, 190)
        self.LableDe.setText('Number of decimals')
        self.LableDe.setFont(self.font1)

        self.Decimal = QSpinBox(self)
        self.Decimal.setObjectName(u"Decimal")
        self.Decimal.setGeometry(QRect(300, 220, 220, 100))
        self.Decimal.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                            "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.Decimal.setMinimum(-5)
        self.Decimal.setMaximum(0)
        self.Decimal.setSingleStep(1)
        self.Decimal.setValue(-2)
        self.Decimal.valueChanged.connect(self.updateDecimal)
        #---------------------------------------------------- Save and close
        self.CancelButton = QPushButton('Cancel', self)
        self.CancelButton.setGeometry(2, 355, 275, 80)
        self.CancelButton.clicked.connect(self.Cancel)

        self.SaveButton = QPushButton("Save", self)
        self.SaveButton.setGeometry(287, 355, 283, 80)
        self.SaveButton.clicked.connect(self.Save)


```

```python

    def Save(self):

        Config.PID_P_x = self.PGain.value()
        Config.PID_D_x = self.DGain.value()
        Config.PID_I_x = self.IGain.value()
        Config.PID_P_y = self.PGain.value()
        Config.PID_D_y = self.DGain.value()
        Config.PID_I_y = self.IGain.value()
        self.hide()

    def Cancel(self):
        self.hide()

    def updateDecimal(self):
        self.PGain.setSingleStep(10 ** (self.Decimal.value()))
        self.DGain.setSingleStep(10 ** (self.Decimal.value()))
        self.IGain.setSingleStep(10 ** (self.Decimal.value()))

#------------------------------------------------------------------ State space config window
class SS_Config(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(700,350,570,440)
        self.setWindowTitle('State Space Config')
        #--------------------------------------------------------------
        self.font = QFont()
        self.font.setPointSize(16)

        self.font2 = QFont()
        self.font2.setPointSize(12)
        # ------------------------------------------------------- info
        self.Info = QLabel(self)
        self.Info.move(10,5)
        self.Info.setText('Change K1 and K2 Gains. Save and rerun the StateSpace')
```

```
        self.Info.setFont(self.font)

        self.info2 = QLabel(self)
        self.info2.move(10, 335)
        self.info2.setText('NB! all values are multiplied by 10')
        self.info2.setFont(self.font2)
        #-------------------------------------------------------- Set K1 value
        self.LableK1 = QLabel(self)
        self.LableK1.move(65,45)
        self.LableK1.setText('K1 Gain')
        self.LableK1.setFont(self.font)

        self.k1Gain = QDoubleSpinBox(self)
        self.k1Gain.setObjectName(u"doubleSpinBox")
        self.k1Gain.setGeometry(QRect(5, 80, 220, 100))
        self.k1Gain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                      "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.k1Gain.setDecimals(5)
        self.k1Gain.setMinimum(0.00000)
        self.k1Gain.setSingleStep(0.01000)
        # -------------------------------------------------------- Set K2 value
        self.LableK1 = QLabel(self)
        self.LableK1.move(360,45)
        self.LableK1.setText('K2 Gain')
        self.LableK1.setFont(self.font)

        self.k2Gain = QDoubleSpinBox(self)
        self.k2Gain.setObjectName(u"doubleSpinBox")
        self.k2Gain.setGeometry(QRect(300, 80, 220, 100))
        self.k2Gain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                      "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.k2Gain.setDecimals(5)
        self.k2Gain.setMinimum(0.00000)
        self.k2Gain.setSingleStep(0.01000)
```

```
        #------------------------------------------------------ Set Decimal
        self.LableK1 = QLabel(self)
        self.LableK1.move(5, 190)
        self.LableK1.setText('Number of decimals')
        self.LableK1.setFont(self.font)

        self.Decimal = QSpinBox(self)
        self.Decimal.setObjectName(u"Decimal")
        self.Decimal.setGeometry(QRect(5, 220, 220, 100))
        self.Decimal.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                           "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.Decimal.setMinimum(-5)
        self.Decimal.setMaximum(0)
        self.Decimal.setSingleStep(1)
        self.Decimal.setValue(-2)
        self.Decimal.valueChanged.connect(self.updateDecimal)
        #---------------------------------------------------- Save and close
        self.SaveButton = QPushButton("Save", self)
        self.SaveButton.setGeometry(287,355,283,80)
        self.SaveButton.clicked.connect(self.Save)
        #----------------------------------------------------Cancel
        self.CancelButton = QPushButton("Cancel",self)
        self.CancelButton.setGeometry(2, 355, 275, 80)
        self.CancelButton.clicked.connect(self.Cancel)

    def Save(self):
        Config.SS_k1_x = self.k1Gain.value()*10
        Config.SS_k2_x = self.k2Gain.value()*10
        Config.SS_k1_y = self.k1Gain.value()*10
        Config.SS_k2_y = self.k2Gain.value()*10
        self.hide()

    def Cancel(self):
        self.hide()

```

```
    def updateDecimal(self):
        self.k1Gain.setSingleStep(10**(self.Decimal.value()))
        self.k2Gain.setSingleStep(10**(self.Decimal.value()))
#---------------------------------------------------------------------- MPC config window

class MPC_Config(QWidget):
    def __init__(self):
        super().__init__()
        self.Decimal = None
        self.setGeometry(700,350,570,440)
        self.setWindowTitle('MPC Config')

        self.font = QFont()
        self.font.setPointSize(16)

        self.font2 = QFont()
        self.font2.setPointSize(12)
        #---------------------------------------------------------- info
        self.Info = QLabel(self)
        self.Info.move(10,5)
        self.Info.setText('Change the Q and R matrix. Save and rerun the MPC')
        self.Info.setFont(self.font)

        self.Info2 = QLabel(self)
        self.Info2.move(10, 335)
        self.Info2.setText('NB! all values are multiplied by 10')
        self.Info2.setFont(self.font2)

        #---------------------------------------------------------- Set K1 value
        self.LableK1 = QLabel(self)
        self.LableK1.move(65, 45)
        self.LableK1.setText('Q Position')
        self.LableK1.setFont(self.font)

        self.k1Gain = QDoubleSpinBox(self)
        self.k1Gain.setObjectName(u"doubleSpinBox")
```

```
        self.k1Gain.setGeometry(QRect(5, 80, 220, 100))
        self.k1Gain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                                  "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.k1Gain.setDecimals(4)
        self.k1Gain.setMinimum(0.00000)
        self.k1Gain.setSingleStep(0.01000)
        # ------------------------------------------------------- Set K2 value
        self.LableK2 = QLabel(self)
        self.LableK2.move(360, 45)
        self.LableK2.setText('Q Velocity')
        self.LableK2.setFont(self.font)

        self.k2Gain = QDoubleSpinBox(self)
        self.k2Gain.setObjectName(u"doubleSpinBox")
        self.k2Gain.setGeometry(QRect(300, 80, 220, 100))
        self.k2Gain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                                  "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.k2Gain.setDecimals(4)
        self.k2Gain.setMinimum(0.00000)
        self.k2Gain.setSingleStep(0.01000)
        # ------------------------------------------------------- Set K3 value
        self.LableK3 = QLabel(self)
        self.LableK3.move(75, 190)
        self.LableK3.setText('R Angle')
        self.LableK3.setFont(self.font)

        self.k3Gain = QDoubleSpinBox(self)
        self.k3Gain.setObjectName(u"doubleSpinBox")
        self.k3Gain.setGeometry(QRect(5, 220, 220, 100))
        self.k3Gain.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
```

```
                                    "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.k3Gain.setDecimals(4)
        self.k3Gain.setMinimum(0.00000)
        self.k3Gain.setSingleStep(0.01000)
        #------------------------------------------------------- Set Decimal
        self.LableDesi = QLabel(self)
        self.LableDesi.move(300, 190)
        self.LableDesi.setText('Number of decimals')
        self.LableDesi.setFont(self.font)

        self.Decimal = QSpinBox(self)
        self.Decimal.setObjectName(u"Decimal")
        self.Decimal.setGeometry(QRect(300, 220, 220, 100))
        self.Decimal.setStyleSheet("QAbstractSpinBox::up-button { width: 100px; height: 50px; }"
                                    "QAbstractSpinBox::down-button { width: 100px; height: 50px; }")
        self.Decimal.setMinimum(-5)
        self.Decimal.setMaximum(0)
        self.Decimal.setSingleStep(1)
        self.Decimal.setValue(-2)
        self.Decimal.valueChanged.connect(self.updateDecimal)
        #--------------------------------------------------- Save and close
        self.SaveButton = QPushButton("Save", self)
        self.SaveButton.setGeometry(287, 355, 283, 80)
        self.SaveButton.clicked.connect(self.Save)
        # ---------------------------------------------------Cancel
        self.CancelButton = QPushButton("Cancel", self)
        self.CancelButton.setGeometry(2, 355, 275, 80)
        self.CancelButton.clicked.connect(self.Cancel)

    def Cancel(self):
        self.hide()

    def Save(self):
```

```
        Config.Log.append('New values saved for the MPC')
        Config.MPC_Q_xp = round(self.k1Gain.value()*10, 5)
        Config.MPC_Q_xv = round(self.k2Gain.value()*10, 5)
        Config.MPC_R_Vin = round(self.k3Gain.value()*10, 5)
        self.hide()

    def updateDecimal(self):
        self.k1Gain.setSingleStep(10 ** (self.Decimal.value()))
        self.k2Gain.setSingleStep(10 ** (self.Decimal.value()))
        self.k3Gain.setSingleStep(10 ** (self.Decimal.value()))


#--------------------------------------------------------------------------------Main GUI

class GUI(QMainWindow):
    def __init__(self):
        super(GUI, self).__init__()

        self.PID_config = PID_Config()
        self.ss_config = SS_Config()
        self.MPC_config = MPC_Config()

        self.setWindowTitle("3 DOF")
        self.setGeometry(0,0, 1280, 800)

        self.circle_rect = QRect(QPoint(50, 80), QPoint(750, 750))
        self.Target_ball_pos = None
        self.FB_ball = None
        self.log_string = []
        self.log_string_old = []
        self.font1 = QFont()
        self.font1.setPointSize(9)

        #---------------------------------------------------------------------- Update Clock
        timer = QTimer(self)
        timer.timeout.connect(self.UpdateInformativeValues)
```

```
        timer.start(550)
        timer1 = QTimer(self)
        timer1.timeout.connect(self.UpdateFBBall)
        timer1.start(100)

        self.ExitButton = QPushButton('Exit', self)
        self.ExitButton.setGeometry(10, 10, 80, 60)
        self.ExitButton.setStyleSheet("background-color: red")
        self.ExitButton.clicked.connect(self.CloseButton)
        #------------------------------------------------------------------ CPU Tmp

        self.CPUTmpLabel = QLabel("CPU Temp:  Crit at < 110",self)
        self.CPUTmpLabel.setGeometry(780,425,150,22)

        self.CPUTmp1 = QLabel(self)
        self.CPUTmp1.setGeometry(780,450,60,22)
        self.CPUTmp1.setFont(self.font1)
        self.CPUTmp1.setFrameShape(QFrame.Panel)

        self.CPUTmp2 = QLabel(self)
        self.CPUTmp2.setGeometry(850, 450, 60, 22)
        self.CPUTmp2.setFont(self.font1)
        self.CPUTmp2.setFrameShape(QFrame.Panel)
        # ----------------------------------------------------------------- box for buttons
        self.ControlerBox = QFrame(self)
        self.ControlerBox.setGeometry(660,105,580,60)
        self.ControlerBox.setFrameShape(QFrame.StyledPanel)
        self.ControlerBox.setLineWidth(2)

        #------------------------------------------------------------------ Controller selector
        self.ControlModeLable = QLabel(self)
        self.ControlModeLable.setGeometry(660, 85, 180, 20)
        self.ControlModeLable.setText("Controller settings: ")

```

```
        self.Controller = QComboBox(self)
        self.Controller.setFont(self.font1)
        self.Controller.addItem("No Control Mode selected")
        self.Controller.addItem("PID")
        self.Controller.addItem("StateSpace")
        self.Controller.addItem("MPC")
        self.Controller.setObjectName(u"comboBox1")
        self.Controller.setGeometry(QRect(670, 110, 190, 50))
        self.Controller.activated[str].connect(self.ControllerSelecter)
        # --------------------------------------------------------------------- Edit controller
        self.EditButton = QPushButton("...", self)
        self.EditButton.setGeometry(870, 110, 40, 50)
        self.EditButton.clicked.connect(self.toggle_window)
        #------------------------------------------------------------------- Input selector

        self.InputMode = QComboBox(self)
        self.InputMode.addItem("No Input selected")
        self.InputMode.addItem("Hunt (not implemented)")
        self.InputMode.addItem("Track (not implemented)")
        self.InputMode.setObjectName(u"comboBox2")
        self.InputMode.setGeometry(QRect(920, 110, 120, 50))
        self.InputMode.activated[str].connect(self.InputtModeSelector)

        # ------------------------------------------------------------------- Start Controller
        self.StartButton = QPushButton("Start", self)
        self.StartButton.setGeometry(1060, 110, 50, 50)
        self.StartButton.setCheckable(True)
        self.StartButton.setStyleSheet("background-color: red")
        self.StartButton.clicked.connect(self.StartButton1)
        self.StartButton.clicked.connect(self.on_button_clicked)

        # ------------------------------------------------------------------- Record data
        self.RecordData = QPushButton('Record Data', self)
```

```
        self.RecordData.setGeometry(1130, 110, 100, 50)
        self.RecordData.setCheckable(True)
        self.RecordData.setStyleSheet("background-color: red")
        self.RecordData.clicked.connect(self.on_button_clicked)
        self.RecordData.clicked.connect(self.StartValuesRecord)
        self.RecordData.pressed.connect(self.MPC_Snap_Shot_True)
        self.RecordData.released.connect(self.MPC_Snap_Shot_False)

        # ---------------------------------------------------------------------- Box for camera buttons and lable
        self.CameraBoxLable = QLabel(self)
        self.CameraBoxLable.setGeometry(590, 2, 150, 12)
        self.CameraBoxLable.setText('Camera Buttons:')

        self.CameraBox = QFrame(self)
        self.CameraBox.setGeometry(590, 15, 280, 60)
        self.CameraBox.setFrameShape(QFrame.StyledPanel)
        self.CameraBox.setLineWidth(2)

        # ------------------------------------------------------------------------ Start Camera
        self.StartCamera = QPushButton(self)
        self.StartCamera.setGeometry(600, 20, 50, 50)
        self.StartCamera.setCheckable(True)
        self.StartCamera.setStyleSheet("background-color: red")
        self.StartCamera.setIcon(QIcon('icons/Camera.svg.png'))
        self.StartCamera.clicked.connect(self.on_button_clicked)
        self.StartCamera.clicked.connect(self.CameraStart)

        self.CamerashowFrame = QPushButton('Frame', self)
        self.CamerashowFrame.setGeometry(670, 20, 50, 50)
        self.CamerashowFrame.setCheckable(True)
        self.CamerashowFrame.setStyleSheet("background-color: red")
        self.CamerashowFrame.clicked.connect(self.on_button_clicked)
        self.CamerashowFrame.clicked.connect(self.CameraShowFrame)

```

```
        self.CamerashowMask = QPushButton('Mask', self)
        self.CamerashowMask.setGeometry(740, 20, 50, 50)
        self.CamerashowMask.setCheckable(True)
        self.CamerashowMask.setStyleSheet("background-color: red")
        self.CamerashowMask.clicked.connect(self.on_button_clicked)
        self.CamerashowMask.clicked.connect(self.CameraShowMask)

        self.CameraStopReading = QPushButton(self)
        self.CameraStopReading.setGeometry(810, 20, 50, 50)
        self.CameraStopReading.setCheckable(True)
        self.CameraStopReading.setChecked(True)
        self.CameraStopReading.setStyleSheet("background-color: green")
        self.CameraStopReading.setIcon(QIcon('icons/Play_Pause.svg.png'))
        self.CameraStopReading.clicked.connect(self.on_button_clicked)
        self.CameraStopReading.clicked.connect(self.PauseAndSetCameraValues)

        # ----------------------------------------------------------------- Arduino box and lable

        self.ArduinoBoxLable = QLabel(self)
        self.ArduinoBoxLable.setGeometry(980, 2, 150, 12)
        self.ArduinoBoxLable.setText('Arduino Buttons:')

        self.ArduinoBox = QFrame(self)
        self.ArduinoBox.setGeometry(980, 15, 240, 60)
        self.ArduinoBox.setFrameShape(QFrame.StyledPanel)
        self.ArduinoBox.setLineWidth(2)

        # ----------------------------------------------------------------- Serial
        self.SerialConnect = QPushButton('Serial', self)
        self.SerialConnect.setGeometry(990, 20, 50, 50)
        self.SerialConnect.setCheckable(True)
        self.SerialConnect.setStyleSheet("background-color: red")
        self.SerialConnect.clicked.connect(self.on_button_clicked)
        self.SerialConnect.clicked.connect(self.StartSerialCom)

```

```
        # ------------------------------------------------------------------ Pause
        self.PausReadWriteConnect = QPushButton(self)
        self.PausReadWriteConnect.setGeometry(1060, 20, 50, 50)
        self.PausReadWriteConnect.setCheckable(True)
        self.PausReadWriteConnect.setChecked(True)
        self.PausReadWriteConnect.setStyleSheet("background-color: green")
        self.PausReadWriteConnect.setIcon(QIcon('icons/Play_Pause.svg.png'))
        self.PausReadWriteConnect.clicked.connect(self.on_button_clicked)
        self.PausReadWriteConnect.clicked.connect(self.PauseAndSetArduinoValues)

        # ------------------------------------------------------------------ Calibrate

        self.Calibrate = QPushButton('Calibrate', self)
        self.Calibrate.setGeometry(1130, 20, 80, 50)
        self.Calibrate.setCheckable(False)
        self.Calibrate.pressed.connect(self.CalibrateArduino_True)
        self.Calibrate.released.connect(self.CalibrateArduino_False)


        # ------------------------------------------------------------------ Stepper
    feedback angles

        self.Stepper1_lable = QLabel('Stepper 1', self)
        self.Stepper1_lable.setFont(self.font1)
        self.Stepper1_lable.setGeometry(335, 0, 100, 26)

        self.Stepper1_TargetValue   = QLabel(self)
        self.Stepper1_TargetValue.setFont(self.font1)
        self.Stepper1_TargetValue.setGeometry(335, 22, 160, 26)
        self.Stepper1_TargetValue.setFrameShape(QFrame.Panel)

        self.Stepper1_FeedbackValue = QLabel(self)
        self.Stepper1_FeedbackValue.setFont(self.font1)
        self.Stepper1_FeedbackValue.setGeometry(335, 50, 160, 26)
        self.Stepper1_FeedbackValue.setFrameShape(QFrame.Panel)

```

```
        self.Stepper2_lable = QLabel('Stepper 2', self)
        self.Stepper2_lable.setFont(self.font1)
        self.Stepper2_lable.setGeometry(660, 678, 100, 26)

        self.Stepper2_TargetValue = QLabel(self)
        self.Stepper2_TargetValue.setFont(self.font1)
        self.Stepper2_TargetValue.setGeometry(660, 700, 160, 26)
        self.Stepper2_TargetValue.setFrameShape(QFrame.Panel)

        self.Stepper2_FeedbackValue = QLabel(self)
        self.Stepper2_FeedbackValue.setFont(self.font1)
        self.Stepper2_FeedbackValue.setGeometry(660, 728, 160, 26)
        self.Stepper2_FeedbackValue.setFrameShape(QFrame.Panel)

        self.Stepper3_lable = QLabel('Stepper 3', self)
        self.Stepper3_lable.setFont(self.font1)
        self.Stepper3_lable.setGeometry(10, 678, 100, 26)

        self.Stepper3_TargetValue = QLabel(self)
        self.Stepper3_TargetValue.setFont(self.font1)
        self.Stepper3_TargetValue.setGeometry(10, 700, 160, 26)
        self.Stepper3_TargetValue.setFrameShape(QFrame.Panel)

        self.Stepper3_FeedbackValue = QLabel(self)
        self.Stepper3_FeedbackValue.setFont(self.font1)
        self.Stepper3_FeedbackValue.setGeometry(10, 728, 160, 26)
        self.Stepper3_FeedbackValue.setFrameShape(QFrame.Panel)

        # ---------------------------------------------------------------- Information

        self.TargetBoxLable = QLabel(self)
        self.TargetBoxLable.setGeometry(760, 172, 100, 12)
        self.TargetBoxLable.setText('Values: ')

        self.TargetBox = QFrame(self)
        self.TargetBox.setGeometry(760, 190, 500, 155)
```

```
        self.TargetBox.setFrameShape(QFrame.StyledPanel)
        self.TargetBox.setLineWidth(2)

        self.PositionX = QLabel(self)
        self.PositionX.setGeometry(765,215,150,20)
        self.PositionX.setText('Position x  : ')

        self.PositionY = QLabel(self)
        self.PositionY.setGeometry(765, 235, 150, 20)
        self.PositionY.setText('Position y  : ')

        self.VelX = QLabel(self)
        self.VelX.setGeometry(765, 255, 150, 20)
        self.VelX.setText('Velocity x  :')

        self.VelY = QLabel(self)
        self.VelY.setGeometry(765, 275, 150, 20)
        self.VelY.setText('Velocity y  :')

        self.AngleX = QLabel(self)
        self.AngleX.setGeometry(765, 295, 150, 20)
        self.AngleX.setText('Angle x     :')

        self.AngleY = QLabel(self)
        self.AngleY.setGeometry(765, 315, 150, 20)
        self.AngleY.setText('Angle y     :')

        #Target Column
        #Title
        self.TargetValues = QLabel(self)
        self.TargetValues.setGeometry(865, 190, 150, 20)
        self.TargetValues.setText('Target Values:')

        self.TargetPosX = QLabel(self)
        self.TargetPosX.setGeometry(895,215,150,20)

```

```
        self.TargetPosY = QLabel(self)
        self.TargetPosY.setGeometry(895, 235, 150, 20)

        self.TargetVelocityY = QLabel(self)
        self.TargetVelocityY.setGeometry(895, 255, 150, 20)

        self.TargetVelocityX = QLabel(self)
        self.TargetVelocityX.setGeometry(895, 275, 150, 20)

        self.TargetAngleX = QLabel(self)
        self.TargetAngleX.setGeometry(895, 295, 150, 20)

        self.TargetAngleY = QLabel(self)
        self.TargetAngleY.setGeometry(895, 315, 150, 20)

        #Feedback Column
        #Title
        self.FeedbackValues = QLabel(self)
        self.FeedbackValues.setGeometry(985, 190, 150, 20)
        self.FeedbackValues.setText('Feedback Values:')

        self.FeedbackPosX = QLabel(self)
        self.FeedbackPosX.setGeometry(1020, 215, 150, 20)

        self.FeedbackPosY = QLabel(self)
        self.FeedbackPosY.setGeometry(1020, 235, 150, 20)

        self.FeedbackVelocityY = QLabel(self)
        self.FeedbackVelocityY.setGeometry(1020, 255, 150, 20)

        self.FeedbackVelocityX = QLabel(self)
        self.FeedbackVelocityX.setGeometry(1020, 275, 150, 20)

        self.FeedbackAngleX = QLabel(self)
        self.FeedbackAngleX.setGeometry(1020, 295, 150, 20)

```

```
        self.FeedbackAngleY = QLabel(self)
        self.FeedbackAngleY.setGeometry(1020, 315, 150, 20)

        # Error Column
        # Title
        self.ErrorValues = QLabel(self)
        self.ErrorValues.setGeometry(1140, 190, 150, 20)
        self.ErrorValues.setText('Error:')

        self.ErrorPosX = QLabel(self)
        self.ErrorPosX.setGeometry(1155, 215, 150, 20)

        self.ErrorPosY = QLabel(self)
        self.ErrorPosY.setGeometry(1155, 235, 150, 20)

        self.ErrorVelocityY = QLabel(self)
        self.ErrorVelocityY.setGeometry(1155, 255, 150, 20)

        self.ErrorVelocityX = QLabel(self)
        self.ErrorVelocityX.setGeometry(1155, 275, 150, 20)

        self.ErrorAngleX = QLabel(self)
        self.ErrorAngleX.setGeometry(1155, 295, 150, 20)

        self.ErrorAngleY = QLabel(self)
        self.ErrorAngleY.setGeometry(1155, 315, 150, 20)

        #------------------------------------------------------- Hz
        self.Hzbox = QFrame(self)
        self.Hzbox.setGeometry(760, 375, 500, 30)
        self.Hzbox.setFrameShape(QFrame.StyledPanel)
        self.Hzbox.setLineWidth(2)

        self.HzLable1 = QLabel('GUI Hz', self)
        self.HzLable1.move(790,350)

```

```
        self.GUIHz = QLabel(self)
        self.GUIHz.setGeometry(800, 380, 100, 20)

        self.HzLable2 = QLabel('Memory Hz', self)
        self.HzLable2.move(870, 350)

        self.SharedMemHz = QLabel(self)
        self.SharedMemHz.setGeometry(880, 380, 150, 20)

        self.HzLable3 = QLabel('BackEnd Hz', self)
        self.HzLable3.move(950, 350)

        self.BackEndHz = QLabel(self)
        self.BackEndHz.setGeometry(960, 380, 150, 20)

        self.HzLable4 = QLabel('Uart Hz', self)
        self.HzLable4.move(1030, 350)

        self.UartComHz = QLabel(self)
        self.UartComHz.setGeometry(1040, 380, 150, 20)

        self.HzLable5 = QLabel('Camera Hz', self)
        self.HzLable5.move(1100, 350)

        self.CameraHz = QLabel(self)
        self.CameraHz.setGeometry(1120, 380, 150, 20)

        self.HzLable6 = QLabel('Controller Hz', self)
        self.HzLable6.move(1180, 350)

        self.ControllerHz = QLabel(self)
        self.ControllerHz.setGeometry(1200, 380, 150, 20)

        # ---------------------------------------------------------------------- Log Box

        self.LogBoxLable = QLabel(self)
```

```
        self.LogBoxLable.setGeometry(970,430, 160,20)
        self.LogBoxLable.setText('Information Box:')

        self.LogBox = QTextEdit(self)
        self.LogBox.setGeometry(970, 450, 300, 320)
        # ------------------------------------------------------------------


    def on_button_clicked(self):
        sender = self.sender()
        if sender.isChecked():
            sender.setStyleSheet("background-color: green")
        else:
            sender.setStyleSheet("background-color: red")

    # ------------------------------------------------------------------ Update
    Informative Values

    def UpdateInformativeValues(self):
        self.Stepper1_TargetValue.setText('Target Angle     : '+ str(round(Config.
    Stepper1_Target,3)))
        self.Stepper1_FeedbackValue.setText('Feedback Angle : '+ str(round(Config.
    Stepper1_Feedback,3)))

        self.Stepper2_TargetValue.setText('Target Angle     : ' + str(round(Config.
    Stepper2_Target,3)))
        self.Stepper2_FeedbackValue.setText('Feedback Angle : ' + str(round(Config.
    Stepper2_Feedback,3)))

        self.Stepper3_TargetValue.setText('Target Angle     : ' + str(round(Config.
    Stepper3_Target,3)))
        self.Stepper3_FeedbackValue.setText('Feedback Angle : ' + str(round(Config.
    Stepper3_Feedback,3)))

        self.TargetPosX.setText(str(Config.Target_Pos_x))
        self.TargetPosY.setText(str(Config.Target_Pos_y))
```

```
        self.TargetVelocityY.setText(str(Config.Target_Vel_x))
        self.TargetVelocityX.setText(str(Config.Target_Vel_y))
        self.TargetAngleX.setText(str(round(Config.U_rad_x/RtD, 3)))
        self.TargetAngleY.setText(str(round(Config.U_rad_y/RtD,3)))

        self.FeedbackPosX.setText(str(Config.cam_x))
        self.FeedbackPosY.setText(str(Config.cam_y))
        self.FeedbackVelocityY.setText(str(Config.cam_x_vel))
        self.FeedbackVelocityX.setText(str(Config.cam_y_vel))
        self.FeedbackAngleX.setText(str(Config.U_x_v_fb))
        self.FeedbackAngleY.setText(str(Config.U_y_v_fb))

        self.ErrorPosX.setText(str(round(Config.Target_Pos_x - Config.cam_x,2)))
        self.ErrorPosY.setText(str(round(Config.Target_Pos_y - Config.cam_y,2)))
        self.ErrorVelocityY.setText(str(-Config.cam_x_vel))
        self.ErrorVelocityX.setText(str(-Config.cam_y_vel))
        self.ErrorAngleX.setText(str(round(Config.U_rad_x/RtD - Config.U_x_v_fb,2)))
        self.ErrorAngleY.setText(str(round(Config.U_rad_y/RtD - Config.U_y_v_fb,2)))

        self.GUIHz.setText('X')
        self.SharedMemHz.setText(str(round(Config.SharedMem_Hz,2)))
        self.BackEndHz.setText(str(round(Config.BackEnd_Hz,2)))
        self.UartComHz.setText(str(round(Config.UartCom_Hz,2)))
        self.CameraHz.setText(str(round(Config.Cam_Hz, 1)))
        self.ControllerHz.setText(str(round(Config.Controller_Hz, 1)))
        # ------------------------------------------------------------ CPU TMP
        if platform.system() == 'Linux':
            self.output = subprocess.check_output(['sensors'])
            self.temp1 = self.output.split()[5].decode()
            self.temp2 = self.output.split()[14].decode()
            self.CPUTmp1.setText(self.temp1)
            self.CPUTmp2.setText(self.temp2)
        else:
            self.CPUTmp1.setText('only linux')
            self.CPUTmp2.setText('only linux')

```

```
        #self.FB_ball = QPoint(400 +(700/400)*Config.cam_x, 400  -(700/400)*Config.cam_y)
        #if self.Target_ball_pos is None:
        #     self.Target_ball_pos =  QPoint(400, 400)
        #self.update()

        self.log_string = '\n'.join(Config.Log)

        if self.log_string != self.log_string_old:
            self.LogBox.setText(self.log_string)

        self.log_string_old = self.log_string

        if not Config.Camera_Runnig:
            self.CamerashowMask.setEnabled(False)
            self.CamerashowMask.setChecked(False)
            self.CamerashowMask.setStyleSheet("background-color: red")
            Config.Camera_show_Mask = 0
            self.CamerashowFrame.setEnabled(False)
            self.CamerashowFrame.setChecked(False)
            self.CamerashowFrame.setStyleSheet("background-color: red")
            Config.Camera_show_Frame = 0
        else:
            self.CamerashowMask.setEnabled(True)
            self.CamerashowFrame.setEnabled(True)



    def StartValuesRecord(self):
        if Config.Record_Data == False:
            Config.Record_Data = True
        else:
            Config.Record_Data = False

    def PauseAndSetArduinoValues(self):
        if Config.Paus_New_Arduino_Values == False:
            Config.Paus_New_Arduino_Values = True
```

```
        else:
            Config.Paus_New_Arduino_Values = False
    # ------------------------------------------------------------- Start button
    action
    def StartButton1(self):
        if Config.Start == False:
            Config.Start = True
        else:
            Config.Start = False

    # ------------------------------------------------------------- Camera

    def CameraStart(self):
        if Config.Camera_Start == False:
            Config.Camera_Start = True
        else:
            Config.Camera_Start = False

    def CameraShowFrame(self):
        if Config.Camera_show_Frame == 0:
            Config.Camera_show_Frame = 1
        else:
            Config.Camera_show_Frame = 0

    def CameraShowMask(self):
        if Config.Camera_show_Mask == 0:
            Config.Camera_show_Mask= 1
        else:
            Config.Camera_show_Mask = 0

    def PauseAndSetCameraValues(self):
        if Config.Camera_Pause < 0.0:
            Config.Camera_Pause = 12.34
        else:
            Config.Camera_Pause = -5.67

```

```
    #------------------------------------------------------------------ Start Com
    def StartSerialCom(self):
        if Config.Start_Serial_Com == False:
            Config.Start_Serial_Com = True
        else:
            Config.Start_Serial_Com = False


    def StartI2CCom(self):
        if Config.Start_I2C_Com == False:
            Config.Start_I2C_Com = True
        else:
            Config.Start_I2C_Com = False

    def CalibrateArduino_True(self):
        if Config.Serial_Com_Running and not Config.Running:
            Config.Calibrate_Arduino = 10.56
            Config.Log.append('Calibrate signal')

    def CalibrateArduino_False(self):
        Config.Calibrate_Arduino = -5.65

    def MPC_Snap_Shot_True(self):
        if Config.Running and Config.Control_Mode == 'MPC' and not Config.Record_Data:
            if Config.MPC_SnapShot < 0.0:
                Config.MPC_SnapShot = 10.57
                Config.Log.append('MPC_Snap')

    def MPC_Snap_Shot_False(self):
        Config.MPC_SnapShot = -5.65

    # ----------------------------------------------------------------- Write Control
Mode to config file
    def ControllerSelecter(self, text):
        Config.Control_Mode = text
```

```

    # ---------------------------------------------------------------------------- Write Input Mode to config file
    def InputtModeSelector(self, text):
        Config.Input_Mode = text

    def UpdateFBBall(self):
        self.FB_ball = QPoint(400 + (700 / 400) * Config.cam_x, 400 - (700 / 400) * Config.cam_y)
        if self.Target_ball_pos is None:
            self.Target_ball_pos = QPoint(400, 400)
        self.update()

    def paintEvent(self, event):
        painter = QPainter(self)
        pen = QPen(Qt.black, 2, Qt.SolidLine)
        brush = QColor(0, 0, 0, 0)
        painter.setPen(pen)
        painter.setBrush(brush)
        painter.drawEllipse(self.circle_rect)

        if self.Target_ball_pos:

            brush1 = QColor(255, 0, 0)
            painter.setBrush(brush1)
            painter.drawEllipse(self.Target_ball_pos, 10, 10)
            brush2 = QColor(255, 255, 0)
            painter.setBrush(brush2)
            painter.drawEllipse(self.FB_ball, 5, 5)

    def mousePressEvent(self, event):
        if event.button() == Qt.LeftButton:
            if self.circle_rect.contains(event.pos()):
                Config.Target_Pos_x = round((400/700)*(-400 + event.x()), 1)
                Config.Target_Pos_y = round((400/700)*(400 - event.y()), 1)
                self.Target_ball_pos = event.pos()
```

```
                #self.update()

    def mouseMoveEvent(self, event):

        if self.circle_rect.contains(event.pos()):
            Config.Target_Pos_x = round((400/700)*(-400 + event.x()), 1)
            Config.Target_Pos_y = round((400/700)*(400 - event.y()), 1)
            self.Target_ball_pos = event.pos()
            #self.update()



    def EditConfig(self):
        print('ops')
        self.ss_config.show()
        ControllerCode = Config.Control_Mode + 'Config'

    def toggle_window(self):
        if Config.Control_Mode == 'PID':
            if not self.PID_config.isVisible():
                self.PID_config.setWindowFlags(Qt.WindowStaysOnTopHint)
                self.PID_config.show()
                self.PID_config.PGain.setValue(Config.PID_P_x)
                self.PID_config.DGain.setValue(Config.PID_D_x)
                self.PID_config.IGain.setValue(Config.PID_I_x)

        if Config.Control_Mode == 'StateSpace':
            if not self.ss_config.isVisible():
                self.ss_config.setWindowFlags(Qt.WindowStaysOnTopHint)
                self.ss_config.show()
                self.ss_config.k1Gain.setValue(Config.SS_k1_x/10)
                self.ss_config.k2Gain.setValue(Config.SS_k2_x/10)

        if Config.Control_Mode == 'MPC':
            if not self.MPC_config.isVisible():
                self.MPC_config.setWindowFlags(Qt.WindowStaysOnTopHint)
```

```
                self.MPC_config.show()
                self.MPC_config.k1Gain.setValue(Config.MPC_Q_xp/10)
                self.MPC_config.k2Gain.setValue(Config.MPC_Q_xv/10)
                self.MPC_config.k3Gain.setValue(Config.MPC_R_Vin/10)

    def CloseButton(self):
        # Display a confirmation dialog before quitting the application
        reply = QMessageBox.question(self, 'Confirm Exit', 'Are you sure you want to exit?',
                                     QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

        if reply == QMessageBox.Yes:
            QApplication.closeAllWindows()

    def closeEvent(self, event):
        # Set a flag to signal the threads to exit
        QApplication.closeAllWindows()
        Config.exit_flag = True
```

B.3 CameraCode.py

```
import cv2
import numpy as np
import time
from multiprocessing import shared_memory
import multiprocessing
import Config as C
import platform

PrevT = time.time()

lock = multiprocessing.Lock()

shm = shared_memory.SharedMemory(name=shared_data.name)
```

```
shm_array = np.ndarray(C.Shm_array.shape, dtype=np.float16, buffer=shm.buf)
#shm_array = C.Shm_array
#shm_array = np.zeros(20)

last_values_x = []
last_values_y = []
last_values_v_x = []
last_values_v_y = []



Camera_Pause = -5.67

ang_x = 0
ang_y = 0

# Define the lower and upper bounds of the orange color in HSV format
orange_lower = np.array([0, 88, 85])
orange_upper = np.array([13, 255, 201])
Show_Frame = 0
Show_Mask = 0

Frame = False
Mask = False
gx = 0
gy = 0
gx_vel = 0
gy_vel = 0


gx_prev = 0
gy_prev = 0
```

```
gx_vel_prev = 0.0
gy_vel_prev = 0.0

# Initialize the video stream
if platform.system() == 'Linux':
    cap = cv2.VideoCapture(0)
elif platform.system() == 'Windows':
    cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)

#cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)
#cap = cv2.VideoCapture(0)


# Set the video resolution to be square, centered on (0,0)
width = 640
height = 480

cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

def average_filter_x(value):
    last_values_x.append(value)
    if len(last_values_x) > 3:
        last_values_x.pop(0)
    return np.average(last_values_x)

def average_filter_y(value):
    last_values_y.append(value)
    if len(last_values_y) > 3:
        last_values_y.pop(0)
    return np.average(last_values_y)




while True:
```

```
    # Read a frame from the video stream
    ret, frame1 = cap.read()

    if ret == False:
        lock.acquire()
        shm_array[4] = round(1, 0)
        shm_array[5] = round(9, 0)
        shm_array[6] = round(9, 0)
        shm_array[7] = round(9, 0)
        lock.release()
        break
    # Crop frame to only include platform
    frame2 = np.zeros((480, 640, 3), dtype=np.uint8)
    cv2.circle(frame2, (320, 240), 205, (255, 255, 255), -1)
    frame = cv2.bitwise_and(frame2, frame1)

    CurrT = time.time()
    dt = CurrT - PrevT
    PrevT = CurrT
    if dt == 0:
        Hz = 9999.99
    else:
        Hz = 1 / dt

    # Convert the frame from BGR color space to HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Threshold the image to isolate the orange color
    mask = cv2.inRange(hsv, orange_lower, orange_upper)

    # Find the contours in the mask
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # If a contour is found, get its center and draw a circle around it
    if len(contours) > 0:
```

```
        c = max(contours, key=cv2.contourArea)
        ((x, y), radius) = cv2.minEnclosingCircle(c)
        if radius > 10:
            cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
            cv2.circle(frame, (int(x), int(y)), 2, (0, 255, 255), -1)
            # Scale the coordinate system in terms of the fisheye lens
            lock.acquire()
            ang_x = shm_array[8]
            ang_y = shm_array[9]
            lock.release()

            gx = -(x - width / 2)
            gy = -(y - (height+16) / 2)

            #gx = -(x - width / 2) / np.cos(ang_x * 2)
            #gy = -(y - (height + 16) / 2) / np.cos(ang_y * 2)

            if -1 < (gx - gx_prev) < 1:
                gx = gx_prev

            if -1 < (gy - gy_prev) < 1:
                gy = gy_prev


            #gx = average_filter_x(gx)
            #gy = average_filter_y(gy)

            if gx_prev == gx:
                gx_vel = 0
            else:
                gx_vel = (gx - gx_prev) / dt
            if gy_prev == gy:
                gy_vel = 0
            else:
                gy_vel = (gy - gy_prev) / dt

```

```
        gx_prev = gx
        gy_prev = gy
else:
    gx = 0
    gy = 0
    gx_vel = 0
    gy_vel = 0

if Camera_Pause > 0.0:
    gx = 0
    gy = 0
    gx_vel = 0
    gy_vel = 0

lock.acquire()
shm_array[4] = round(gx, 0)
shm_array[5] = round(gy, 0)
shm_array[6] = round(gx_vel, 0)
shm_array[7] = round(gy_vel, 0)
Show_Frame    = int(shm_array[21])
Show_Mask     = int(shm_array[22])
Camera_Pause =shm_array[23]
shm_array[24] = Hz
lock.release()


if Show_Frame:
    cv2.imshow("frame", frame)
    Frame = True
elif (not Show_Frame and Frame):
    cv2.destroyWindow("frame")
    Frame = False

if Show_Mask:
    cv2.imshow("mask", mask)
    Mask = True
```

```
    elif (not Show_Mask and Mask):
        cv2.destroyWindow("mask")
        Mask = False

    # Wait for a key press and exit if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video stream and close all windows
cap.release()
cv2.destroyAllWindows()
```

B.4 MPC.py

```
import time
import pickle
import platform

if platform.system() == 'Linux':
    print('import Mpc_Linux')
    from MPC_code_Linux_05.cpg_solver import cpg_solve
elif platform.system() == 'Windows':
    from MPC_code.cpg_solver import cpg_solve

from multiprocessing import shared_memory
import numpy as np
import multiprocessing
import Config
import datetime

u_traj = np.array([0, 0])

if platform.system() == 'Linux':
    print('open Mpc_Linux')
    with open('MPC_code_Linux_05/problem.pickle', 'rb') as f:
```

```
        problem = pickle.load(f)
elif platform.system() == 'Windows':
    with open('MPC_code/problem.pickle', 'rb') as f:
        problem = pickle.load(f)

lock = multiprocessing.Lock()
shm = shared_memory.SharedMemory(name=shared_data.name)
shm_array = np.ndarray(Config.Shm_array.shape, dtype=np.float16, buffer=shm.buf)


# Assign Parameters for MPC
Apar = np.array([[0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 1.0], [0.0,
    0.0, 0.0, 0.0]])
Bpar = np.array([[0.0, 0.0], [7000.0, 0.0], [0.0, 0.0], [0.0, 7000.0]])

problem.param_dict['A'].value = Apar
problem.param_dict['B'].value = Bpar


x_pos_prev = 0
y_pos_prev = 0
x_vel = 0
y_vel = 0
t3 = 0


PrevT = -0.001

lock.acquire()
MPC_Q_xp = shm_array[15]
MPC_Q_xv = shm_array[16]
MPC_R_Vin = shm_array[17]
lock.release()

MPC_q = np.diag([MPC_Q_xp, MPC_Q_xv, MPC_Q_xp, MPC_Q_xv])
MPC_r = np.diag([MPC_R_Vin, MPC_R_Vin])
```

```
problem.param_dict['Qsqrt'].value = MPC_q
problem.param_dict['Rsqrt'].value = MPC_r
SnapShotRunnig = False
SnapTaken = False
WriteSnap = False
TimeAdded = False
path = 'SavedData/Snap/'
sine_out = 0
while True:

    CurrT = time.time()
    dt = CurrT - PrevT
    PrevT = CurrT
    if dt == 0:
        Hz = 9999.99
    else:
        Hz = round(1/dt, 2)
    #print(dt)

    lock.acquire()
    #sine = shm_array[2]
    #shm_array[3] = sine_out
    x_pos           = shm_array[4]
    y_pos           = shm_array[5]
    x_vel           = shm_array[6]
    y_vel           = shm_array[7]
    s_r             = np.array([[shm_array[0]], [0], [shm_array[1]], [0]])
    shm_array[8]    = round(u_traj[0], 4)
    shm_array[9]    = round(u_traj[1], 4)
    shm_array[18]   = Hz
    MPC_SnapShot    = shm_array[19]

    lock.release()

    s_states = np.array([[x_pos], [x_vel], [y_pos], [y_vel]])
```

```

    s_error = (s_r - s_states)# Initial state

    problem.param_dict['s_error'].value = s_error
    problem.register_solve('CPG', cpg_solve)
    problem.solve(method='CPG')
    angle_list = problem.var_dict['U'].value
    state_list = problem.var_dict['S'].value
    u_traj = angle_list[:, 1]
    #sine_out = sine
    # ---------------------------------------------------- Take a snapshot of the predict
    if MPC_SnapShot > 0.0 and not SnapShotRunnig:
        StartTime = time.time()
        FB_U_x = []
        FB_P_x = []
        FB_V_x = []
        FB_U_y = []
        FB_P_y = []
        FB_V_y = []
        TimeList = []
        SnapShotRunnig = True

    if SnapShotRunnig:
        RunTime = time.time() - StartTime
        if RunTime > 1:
            if not SnapTaken:
                Predicted_U_x = angle_list[0, :]
                Predicted_S_P_x = state_list[0, :]
                Predicted_S_V_x = state_list[1, :]
                Predicted_U_y = angle_list[1, :]
                Predicted_S_P_y = state_list[2, :]
                Predicted_S_V_y = state_list[3, :]
                SnapTaken = True
            FB_U_x.append(u_traj[0])
            FB_P_x.append(s_error[0])
```

```
            FB_V_x.append(s_error[1])

            FB_U_y.append(u_traj[1])
            FB_P_y.append(s_error[2])
            FB_V_y.append(s_error[3])
            TimeList.append(RunTime-1)

        if RunTime > 2:
            SnapShotRunnig = False
            WriteSnap = True
            FileCreated = False
            n_predict = 0
            n_feedback = 0

    if WriteSnap:
        if not FileCreated:
            now = datetime.datetime.now()
            start_Record_time = time.time()
            timestamp = now.strftime("%Y-%m-%d_%H-%M-%S")
            filename_Predicted_x = f"{path} MPC SnapShot Predicted x {timestamp}.txt"
            filename_Predicted_y = f"{path} MPC SnapShot Predicted y {timestamp}.txt"
            filename_FB_x = f"{path} MPC SnapShot True x {timestamp}.txt"
            filename_FB_y = f"{path} MPC SnapShot True y {timestamp}.txt"
            FileCreated = True

        if (len(Predicted_U_x) > n_predict):
            with open(filename_Predicted_x, "a") as file_1:
                data1 = f'{Predicted_U_x[n_predict]} {Predicted_S_P_x[n_predict]} {Predicted_S_V_x[n_predict]}'
                file_1.write(data1 + '\n')

            with open(filename_Predicted_y, "a") as file_2:
                data2 = f'{Predicted_U_y[n_predict]} {Predicted_S_P_y[n_predict]} {Predicted_S_V_y[n_predict]}'
                file_2.write(data2 + '\n')

```

```
            n_predict += 1


        if (len(FB_U_x) > n_feedback):
            with open(filename_FB_x, "a") as file_3:
                data3 = f'{FB_U_x[n_feedback]} {FB_P_x[n_feedback]} {FB_V_x[n_feedback]} {TimeList[n_feedback]}'
                file_3.write(data3 + '\n')

            with open(filename_FB_y, "a") as file_4:
                data4 = f'{FB_U_y[n_feedback]} {FB_P_y[n_feedback]} {FB_V_y[n_feedback]} {TimeList[n_feedback]}'
                file_4.write(data4 + '\n')
            n_feedback += 1

        else:
            file_1.close()
            file_2.close()
            file_3.close()
            file_4.close()
            WriteSnap = False
```

B.5 C_CodeGenerator.py

```
import cvxpy as cp
import numpy as np

# define dimensions
H, n, m = 10, 4, 2

# define variables
U = cp.Variable((m, H), name='U')
S = cp.Variable((n, H+1), name='S')

# define parameters
```

```
Q = cp.Parameter((n, n), name='Qsqrt')
R = cp.Parameter((m, m), name='Rsqrt')
A = cp.Parameter((n, n), name='A')
B = cp.Parameter((n, m), name='B')
s_error = cp.Parameter((n, 1), name='s_error')
dt = 0.05


# discrete-time dynamics
Apar = np.array([[0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 1.0], [0.0,
    0.0, 0.0, 0.0]])
Bpar = np.array([[0.0, 0.0], [7000.0, 0.0], [0.0, 0.0], [0.0, 7000.0]])
A.value = Apar
B.value = Bpar

# cost
Q.value = np.diag([1.0, 0.1, 1.0, 0.1])
R.value = np.diag([0.0000, 0.0000])

# measurement
s_error.value = np.array([[20], [0], [20], [0]])


cost = 0.0
constr = []
# define objective
for t in range(H):
    #Make the Cost Function in terms of total error from reference point + control angle
    cost += cp.sum_squares(Q@(S[:, t:t+1]))
    cost += cp.sum_squares(R@(U[:, t]))
    #Update position and velocity states for the next timestep
    constr.append(S[:, t+1] == S[:, t] + dt*(A @ S[:, t] + B@U[:, t]))
    #Constrain the control angle in radians
    constr += [U[:, t] <= 0.43]
    constr += [U[:, t] >= -0.43]
constr += [S[:, 0] == s_error[:, 0]]
```

```

#Solve the problem based on the optimal trajectory and input angle from the MPC
problem = cp.Problem(cp.Minimize(cost), constr)


val = problem.solve()


print(val)
print(U.value)
from cvxpygen import cpg

cpg.generate_code(problem, code_dir='MPC_code')
```

B.6 PID.py

```
from multiprocessing import shared_memory
import numpy as np
import multiprocessing
import Config
import time

PrevT = time.time()

U_x = 0
U_y = 0

integral_x = 0
integral_y = 0

Feedback_x = 0
Feedback_y = 0

PrevError_x = 0
PrevError_y = 0
```

```
RtD = (3.14/180)
Run = True

lock = multiprocessing.Lock()
shm = shared_memory.SharedMemory(name=shared_data.name)
shm_array = np.ndarray(Config.Shm_array.shape, dtype=np.float16, buffer=shm.buf)

lock.acquire()
Pid_p_x = shm_array[10]
Pid_d_x = shm_array[11]
Pid_i_x = shm_array[12]

Pid_p_y = shm_array[10]
Pid_d_y = shm_array[11]
Pid_i_y = shm_array[12]
lock.release()

while Run:

    CurrT = time.time()
    dt = CurrT - PrevT
    PrevT = CurrT
    if dt == 0:
        HZ = 9999.99
    else:
        HZ = round(1 / dt, 2)

    lock.acquire()
    Target_x_p = shm_array[0]
    Target_y_p = shm_array[1]
    Feedback_x_p = shm_array[4]
    Feedback_y_p = shm_array[5]
    shm_array[8] = round(U_x, 4)
    shm_array[9] = round(U_y, 4)
    shm_array[18] = HZ
    # print(shm_array[4])
```

```
    lock.release()



    #------------------------------------------------------------- Pid x
    Error_x = Feedback_x_p - Target_x_p

    DeDt_x = (Error_x-PrevError_x)/dt
    integral_x = integral_x + Error_x*dt

    PrevError_x = Error_x

    U_x = Pid_p_x*Error_x + Pid_d_x*DeDt_x + Pid_i_x*integral_x
    #------------------------------------------------------------- Pid y
    Error_y = Feedback_y_p - Target_y_p

    DeDt_y = (Error_y - PrevError_y) / dt
    integral_y = integral_y + Error_y * dt
    PrevError_y = Error_y

    U_y = Pid_p_y * Error_y + Pid_d_x * DeDt_y + Pid_i_y*integral_y

    print(Pid_p_x*Error_x, Pid_d_x*DeDt_x)

    time.sleep(0.01)
```

B.7 State_Space.py

```
import Config
import time
from multiprocessing import shared_memory
import numpy as np
import multiprocessing

global shm
```

```
global shm_array

PrevT = time.time()

U_x = 0
U_y = 0

SS_k1_x = Config.SS_k1_x
SS_k2_x = Config.SS_k2_x

SS_k1_y = Config.SS_k1_y
SS_k2_y = Config.SS_k2_y

Feedback_x_p = 0
Feedback_y_p = 0
PrevFeedback_x_p = 0
PrevFeedback_y_p = 0

x_v_Error = 0
y_v_Error = 0
Run = True

lock = multiprocessing.Lock()
shm = shared_memory.SharedMemory(name=shared_data.name)
shm_array = np.ndarray(Config.Shm_array.shape, dtype=np.float16, buffer=shm.buf)

lock.acquire()
SS_k1_x = shm_array[13]
SS_k2_x = shm_array[14]

SS_k1_y = shm_array[13]
SS_k2_y = shm_array[14]

lock.release()

while Run:
```

```
    CurrT = time.time()
    dt = CurrT - PrevT
    PrevT = CurrT
    if dt == 0:
        Hz = 9999.99
    else:
        Hz = round(1 / dt, 2)
    lock.acquire()
    Target_x_p       = shm_array[0]
    Target_y_p       = shm_array[1]
    Target_x_v       = shm_array[2]
    Target_y_v       = shm_array[3]
    Feedback_x_p     = shm_array[4]
    Feedback_y_p     = shm_array [5]
    x_v_Error        = shm_array[6]
    y_v_Error        = shm_array[7]
    shm_array[8]     = round(U_x, 4)
    shm_array[9]     = round(U_y, 4)
    shm_array[18] = Hz

    lock.release()

    x_p_Error =  Feedback_x_p - Target_x_p
    U_x = (x_p_Error * SS_k1_x + (x_v_Error - Target_x_v )* SS_k2_x)

    y_p_Error =  Feedback_y_p - Target_y_p
    U_y = (y_p_Error * SS_k1_y + (y_v_Error - Target_y_v) * SS_k2_y)

    PrevFeedback_x_p = Feedback_x_p
    PrevFeedback_y_p = Feedback_y_p

    time.sleep(0.001)
    #print(shm_array, U_x)
```

B.8 ControllerTemplate.py

```
import Config
import time
from multiprocessing import shared_memory
import numpy as np
import multiprocessing

PrevT = time.time()

U_x = 0
U_y = 0




Run = True
lock = multiprocessing.Lock()
shm = shared_memory.SharedMemory(name=shared_data.name)
shm_array = np.ndarray(Config.Shm_array.shape, dtype=np.float16, buffer=shm.buf)

while Run:
    #-------------------------------------------------- HZ
    CurrT = time.time()
    dt = CurrT - PrevT
    PrevT = CurrT
    if dt == 0:
        HZ = 9999.99
    else:
        HZ = 1 / dt
    # print(HZ)
    #--------------------------------------------

    lock.acquire()
    Target_x_p = shm_array[0]
    Target_y_p = shm_array[1]
```

```
    shm_array[2] = round(U_x, 4)
    shm_array[3] = round(U_y, 4)
    print(shm_array)
    lock.release()
```

B.9 Config.py

```
import numpy as np

Control_Mode = 'No controllmode selected'
Input_Mode    = ''

Shm_array = np.zeros(25, dtype=np.float16)

exit_flag = False

Log = []
Log_Rev = []
Start = False
Running = False

Record_Data = False
Record_Data_Running = False
Data_Names = 'TX TY FBX FBY FBVX FBVY TUX TUY FBUX FBUY Time'
Hz_Names = 'BackEnd_Hz SharedMem_Hz UartCom_Hz Cam_Hz Controller_Hz'
Sine_Name = 'sine sine_controller fb1 fb2 fb3 BackEnd_Hz Controller_Hz Time'

Camera_Start = False
Camera_Runnig = False
Camera_show_Frame = 0
Camera_show_Mask = 0
Camera_Pause = -5.67

Target_Pos_x    = 0
Target_Pos_y    = 0
```

```
Target_Vel_x = 0
Target_Vel_y = 0
Feedback_pos = 0

Start_Serial_Com = False
Serial_Com_Running = False

Start_I2C_Com = False
I2C_Com_Running = False

Paus_New_Arduino_Values = False
Calibrate_Arduino = -5.65


cam_x = 0.0
cam_x_vel = 0.0
cam_y = 0.0
cam_y_vel = 0.0
No_Ball = 0.0


Stepper1_Target = 0
Stepper1_Feedback = 0
Stepper2_Target = 0
Stepper2_Feedback = 0
Stepper3_Target = 0
Stepper3_Feedback = 0

U_rad_x = 0
U_rad_y = 0

U_x_v_fb = 0.0
U_y_v_fb = 0.0

#-------------------- PID parameter

```

```
PID_P_x = 0.0015
PID_D_x = 0.0000
PID_I_x = 0

PID_P_y = 0.0015
PID_D_y = 0.0000
PID_I_y = 0

Pid_delay_time = 0.01
#------------------------------ State Space

SS_k1_x = 0.0012
SS_k2_x = 0.0006

SS_k1_y = 0.0012
SS_k2_y = 0.0006

SS_delay_time = 0.01
#----------------------------------- MPC

MPC_SnapShot = -5.67

MPC_Q_xp = 1.73
MPC_Q_xv = 0.55
MPC_R_Vin = 310



#MPC_q = np.diag([MPC_Q_xp, MPC_Q_xv, MPC_Q_xp, MPC_Q_xv])
#MPC_r = np.diag([MPC_R_Vin, MPC_R_Vin])


#--------------------------------------------------------------------- Hz

BackEnd_Hz    = 0.0
SharedMem_Hz  = 0.0
```

```
UartCom_Hz      = 0.0

Cam_Hz          = 0.0
Controller_Hz = 0.0

sine = 0.0
sine_controller = 0.0
sine_fb = 0.0
```

B.10 ArduinoCode

```

#include "FastAccelStepper.h"
#include "AVRStepperPins.h"

#define SW_1         10
#define SW_2         11
#define SW_3         12

#define Pul_1        6
#define Dir_1        51

#define Pul_2        7
#define Dir_2        52

#define Pul_3        8
#define Dir_3        53

int sensorVal_1 = LOW;
int sensorVal_2 = LOW;
int sensorVal_3 = LOW;

int Old_SW1 = LOW;
int Old_SW2 = LOW;
int Old_SW3 = LOW;
```

```

float Calibrate_signal = -5.65;

float Target_1 = 0.0;
float Target_2 = 0.0;
float Target_3 = 0.0;

int Target_1_s = 0;
int Target_2_s = 0;
int Target_3_s = 0;

int Current_1_is = 0;
int Current_2_is = 0;
int Current_3_is = 0;

int Pos_1_s = int(-165*0.78)*2; // 800s
int Pos_2_s = int(-165*0.78)*2; // 800s
int Pos_3_s = int(-165*0.78)*2; // 800s

float DegToStep = 1600/360;

float Current_1_d = 0.0;
float Current_2_d = 0.0;
float Current_3_d = 0.0;
float values[5];

FastAccelStepperEngine engine = FastAccelStepperEngine();
FastAccelStepper *stepper_1 = NULL;
FastAccelStepper *stepper_2 = NULL;
FastAccelStepper *stepper_3 = NULL;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);

  pinMode(SW_1      , INPUT);
```

```
  pinMode(SW_2      , INPUT);
  pinMode(SW_3      , INPUT);

  engine.init();
  stepper_1 = engine.stepperConnectToPin(Pul_1);
  stepper_2 = engine.stepperConnectToPin(Pul_2);
  stepper_3 = engine.stepperConnectToPin(Pul_3);
  stepper_1->setDirectionPin(Dir_1, false);
  stepper_2->setDirectionPin(Dir_2, false);
  stepper_3->setDirectionPin(Dir_3, false);

  stepper_1->setSpeedInHz(2600);       // 500 steps/s
  stepper_1->setAcceleration(10000);    // 100 steps/s^2

  stepper_2->setSpeedInHz(2600);       // 500 steps/s
  stepper_2->setAcceleration(10000);    // 100 steps/s^2

  stepper_3->setSpeedInHz(2600);       // 500 steps/s
  stepper_3->setAcceleration(10000);    // 100 steps/s^2
  Calibrate();
}
//----------------------------------------------------- loop
void loop() {
  UsbCom();

  if (Calibrate_signal >= 0){
    Calibrate();
  }
  sensorVal_1 = digitalRead(SW_1);
  sensorVal_2 = digitalRead(SW_2);
  sensorVal_3 = digitalRead(SW_3);

  if ((sensorVal_1)&& !(Old_SW1)){
       stepper_1->forceStopAndNewPosition(Pos_1_s);
  }
  if ((sensorVal_2)&& !(Old_SW2)){
```

```
      stepper_2->forceStopAndNewPosition(Pos_2_s);

  }
  if ((sensorVal_3)&& !(Old_SW3)){
      stepper_3->forceStopAndNewPosition(Pos_3_s);
  }

  Current_1_is = stepper_1->getCurrentPosition();
  Current_2_is = stepper_2->getCurrentPosition();
  Current_3_is = stepper_3->getCurrentPosition();
  Current_1_d = Current_1_is/DegToStep;
  Current_2_d = Current_2_is/DegToStep;
  Current_3_d = Current_3_is/DegToStep;

  Target_1_s = int(Target_1*DegToStep);
  Target_2_s = int(Target_2*DegToStep);
  Target_3_s = int(Target_3*DegToStep);

  stepper_1->moveTo(Target_1_s);
  stepper_2->moveTo(Target_2_s);
  stepper_3->moveTo(Target_3_s);

  Old_SW1 = sensorVal_1;
  Old_SW2 = sensorVal_2;
  Old_SW3 = sensorVal_3;
}

void UsbCom(){
  if (Serial.available() >= 16) { // Wait for 12 bytes (3 floats)

    Serial.readBytes((char*)values, 16);
    Target_1 = values[0];
    Target_2 = values[1];
    Target_3 = values[2];
    Calibrate_signal = values[3];
    Serial.print(Current_1_d);
```

```
    Serial.print(" ");
    Serial.print(Current_2_d);
    Serial.print(" ");
    Serial.print(Current_3_d);
    Serial.print(" ");
    Serial.println(Calibrate_signal);

  }
}

void Calibrate(){
  bool Start = HIGH;
  while (Start){
    UsbCom();
    sensorVal_1 = digitalRead(SW_1);
    sensorVal_2 = digitalRead(SW_2);
    sensorVal_3 = digitalRead(SW_3);

    if (sensorVal_1 and sensorVal_2 and sensorVal_3){
      Start = LOW;
      break;
    }
    if (!sensorVal_1){
      stepper_1->move(-1);
    }else{
      stepper_1->forceStop();
    }

    if (!sensorVal_2){
      stepper_2->move(-1);
    }else{
      stepper_2->forceStop();
    }

    if (!sensorVal_3){
      stepper_3->move(-1);
```

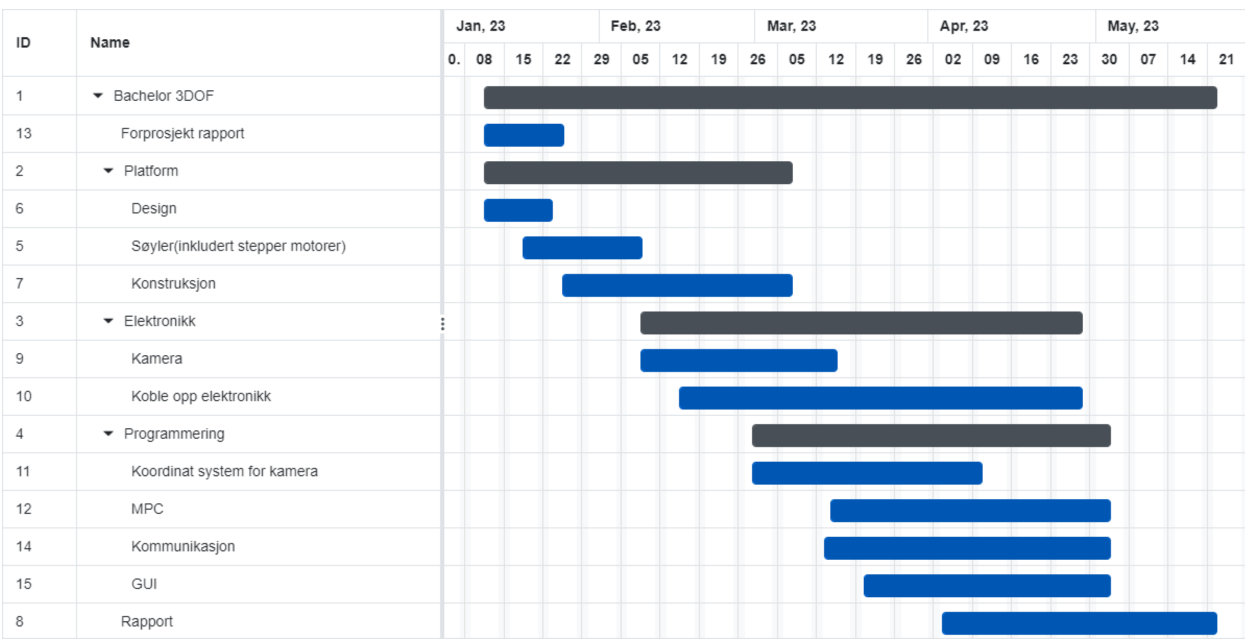
```
    }else{
      stepper_3->forceStop();
    }

  delay(5);
  }
  stepper_1->forceStopAndNewPosition(Pos_1_s);
  stepper_2->forceStopAndNewPosition(Pos_2_s);
  stepper_3->forceStopAndNewPosition(Pos_3_s);
  Target_1_s = 0;
  Target_2_s = 0;
  Target_3_s = 0;
  Calibrate_signal = -5.65;
  delay(1000);
}
```

Appendix C

Gantt Diagram

Gantt Diagram of the progress plan made at the start of the project. Small changes have been made throughout to go into more detail.



Appendix D

Pre-project Report And Progress Reports

This chapter includes the pre-project report and progress reports made throughout the project.

D.1 Pre-project report

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

TITTEL:
3DOF PLATFORM 2.0

KANDIDATNUMMER(E):			
DATO: **11.01**	EMNEKODE: **IE303612**	EMNE: **Bacheloroppgave**	DOKUMENT TILGANG: - Åpen
STUDIUM: **BIELEKTRO**		ANT SIDER/VEDLEGG: 11/0	BIBL. NR: - Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E): **ERLEND COATES (NTNU), AGUS HASAN(NTNU)**

OPPGAVE/SAMMENDRAG:

Utvikle 3DOF plattform som er mer robusto og effektiv, har nye tillegsfunksjoner, og som har nye regulerings-metoder.

Postadresse	Besøksadresse	Telefon	Telefax	Bankkonto
Høgskolen i Ålesund	Larsgårdsvegen 2	70 16 12 00	70 16 13 00	7694 05 00636
N-6025 Ålesund	**Internett**	**Epostadresse**		**Foretaksregisteret**
Norway	www.hials.no	postmottak@hials.no		NO 971 572 140

INNHOLD

1 INNLEDNING

Alle gruppemedlemmene har erfaring fra valgfaget *Kybernetikk* fra forrige semester, og synes at 3DOF-plattformen var spennende fra faget *Dynamiske systemer* i det 4. semesteret. Det å kunne utvikle plattformen og regulere den ved bruk av nye metoder virket som en interessant oppgave. Dessuten har alle gruppemedlemmene erfaring fra *Intelligente Systemer* dersom vi velger å bruke metoder som *fuzzy logic* og *neural network* for å optimalisere reguleringen til plattformen.

2 BEGREPER

-3DOF Platform
3 degrees of freedom platform – en platform som har 3 bevegeøsesretinger: heving/senking (z-aksen), panorering (x-aksen) og kanting (y-aksen)

-PID
PID (Proportional-Integral-Derivative) er en type reguleringsalgoritme som brukes til å holde en variabel (for eksempel en temperatur eller hastighet) innenfor et ønsket settpunkt.

-State Space
State-space control er en metode for å designe kontrollsystemer ved å beskrive systemet i form av en sett med ligninger som beskriver systemets tilstand. Disse ligningene kalles state-space ligninger.

-MPC
MPC (Model Predictive Control) er en metode for å designe kontrollsystemer ved å formulere et optimalt reguleringsproblem og løse dette problemet for å finne den beste kontrollstrategien.

-Step Motor
Step motors er en type elektrisk motor som bruker skrittstyring for å bevege seg. De bruker elektromagnetiske krefter for å skape rotasjon, og kan bevege seg i små, nøyaktige skritt i stedet for å rotere jevnt som en konvensjonell motor.

-Neural Network
Neural network, er en type datamaskinmodell som er inspirert av den biologiske hjernen. Det er et system av algoritmer som er designet for å gjenkjenne mønstre og å lære av erfaring. Ved å gjenkjenne mønster så kan et neuralt nettverk beregne og gi en utgangsvariabel.

-Fuzzy Logic
Fuzzy logic er en metode for å håndtere usikkerhet og subjektive vurderinger ved å bruke grader av sannhet i stedet for enten-eller-sannheter. Det er en form for sannsynlighetslogikk som gir en mer naturlig måte å beskrive og håndtere usikkerhet på. Fuzzy-logikk brukes ofte i kontrollsystemer, der det kan være usikkerhet i inngangene og det er ønskelig å designe en kontrollør som tar hensyn til denne usikkerheten.

-Maskinsyn
Maskinsyn er en teknologi som bruker bildeanalyse og kunstig intelligens for å gi maskiner evnen til å se og forstå visuelle data. Det bruker sensorer, som kameraer, og algoritmer for å analysere og forstå bilder og videoer.

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)
539212 539198 536150

Tabell: Studentnummer(e) for alle i gruppen som leverer oppgaven for bedømmelse i faget ID 302906

3.1.1 Oppgaver for prosjektgruppen – organisering

Fylle inn GANTT-skjema for å logge tidsbruk og koordinere framtidige arbeidsoppgaver for å nå nye milepæler og mål for prosjektet.

3.1.2 Oppgaver for prosjektleder

Ansvarsområde
Følge opp tidsbruk- og sikre et godt samarbeidsmiljø.

Arbeidsoppgaver
Sikre at alle gruppemedlemmer bidrar like mye. Sjekke at progresjonen samsvarer med gantt--skjemaet. Løse konflikter dersom det skulle oppstå.

3.1.3 Oppgaver for sekretær

Ansvarsområde
-Sikre god informasjonsflyt mellom medlemmer og veileder

Arbeidsoppgaver
Skrive referat fra møter-, dokumentere hvilke arbeid som er gjennomført

3.1.4 Oppgaver for øvrige medlem(mer)

-Ansvarsområder
Være et godt gruppemedlem som har en åpen og god dialog.

-Arbeidsoppgaver
Være produktiv og møte til oppsatt tid. Engasjere seg i oppgaven og komme med løsninger og ideèr. Huske å logge tidsbruk på de ulike arbeidsoppgavene, og dokumentere hvordan problem ble løst.

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

4 AVTALER

4.1 Avtale med oppdragsgiver

Forbedre 3DOF Platform brukt i tidligere prosjekt ved å implementere MPC og bedre stabilitet.

Oppdragsgiver/Veileder skal holdes informert om progresjon jevnt gjennom prosjektet. Møter holdes originalt hver andre uke. Bestillinger av deler skal ha klarsignal fra oppdragsgiver/veileder før de er gjennomført.

4.2 Arbeidssted og ressurser

Møtested for arbeid med bacheloroppgaven blir rom L160/L163/L167/Manulab, dette er avhengig av hvilke rom som er tilgjengelig. Spesifikasjoner eller endringer av rom avtales felles med gruppe via Messenger/fysisk dagen før. Ved annet arbeid som ikke krever oppmøte, eller andre årsaker som hindrer medlemmer fra å kunne møte opp, så foregår møtene digitalt via Discord.

Informasjon og filer er tilgjengelige i Microsoft Teams gruppe eller privat Discord tekstkanal tilgjengelig kun for medlemmer.

Ressurser og annet material som er nødvendig til oppgaven avtales med veileder og/eller skaffes av medlemmer etter felles avtale.

4.3 Gruppenormer – samarbeidsregler – holdninger.

- Alle på gruppen skal respektere avtalte arbeidstidpunkter og møter, og møte opp til disse med mindre en god grunn hindrer medlemmet fra dette.
- Beslutninger og endringer skal diskuteres i plenum med alle gruppemedlemmer før en avgjørelse blir tatt.
- Alle gruppemedlemmer har som ansvar og følge med regelmessig i kommunikasjonsmedium som Messenger og Discord slik at beskjeder og møtetidspunkt er oppfattet.
- Alle gruppemedlemmer er pålagt å ha organisert og strukturerte metoder for versjonskontroll, filkontroll, og filnavn av koder og filer som brukes felles.
- Gruppemedlemmer er pålagt å ha positive holdninger 😊

Som utøver av en slik profesjon sikter vi til å kunne levere best mulig resultat på minst mulig tid. Vi håper på gode diskusjoner og arbeidsoppgaver som øker trivselen i gruppa, samtidig som at erfaringene og læringspotensialet til alle medlemmer er gode. Med gode erfaringer har vi gode muligheter til å fortsette med videre prosjekt, og ett godt utgangspunkt for hvordan man skal arbeide felles med andre ingeniører.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Hovedmålet er å implementere en MPC kontroller for å styre en kule på en 3DOF plattform, dette målet kan bli delt opp i mindre delmål som beskrives under.

- Bygge en stabil 3DOF plattform
- Implementere en sensor/sensorpakke som returnerer posisjon og andre ønskelige verdier.
- Lage en matematisk modell for systemet
- Kode en MPC kontroller
- Lage en GUI for å styre posisjon til kulen

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Hovedkravet er å levere en 3DOF platform som kontrolleres ved hjelp av en *Khadas VIM og MPC.* Prosjektet skal resultere i en totalpakke som er enkel og effektiv til undervisning ved et kybernetikk-fag. Dette skal gjøres innenfor budsjettrammene.

Når prosjektet blir levert skal det følge med dokumenter som beskriver hvordan plattformen er konstruert, hvilken software som er brukt, hvordan alt har blitt satt opp, og hvilke metoder som er tatt i bruk for å regulere plattformen. Løs

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Gruppen vil jobbe for å nå planlagte delmål, både individuelt og samlet. Ulike arbeidsoppgaver blir fordelt på de ulike gruppemedlemmene basert på hverandres styrker og erfaringer. Ellers vil gruppen ha en ganske *hands-on* framgangsmåte der man prøver og feiler for å komme seg framover. Svakheten med metoden er at man kan sette seg fast ved et problem, og dermed bruke for lang tid på å løse det, derfor er gruppen klar over at det i visse perioder kan kreve flere arbeidstimer og mindre fritid i løpet av prosjektet.

5.4 Informasjonsinnsamling – utført og planlagt

Gruppen er klar over at det finnes mange løsninger på hvordan 3DOF-platform er bygd opp og hvordan de blir kontrollert ved hjelp av ulike metoder. Dette vil gruppen se nærmere på og kanskje hente inspirasjon i fra. Informasjonen som blir funnet vil bli lagret i Microsoft Teams gruppe eller privat Discord tekstkanal tilgjengelig kun for medlemmer.

5.5 Vurdering – analyse av risiko

Det som er særlig viktig for kunne lykkes er at bestillinger og beslutninger tas raskt og ved første mulighet. Dette er slik at vi hindrer mest mulig dødtid og får mest mulig tid til å jobbe med hoved oppgavene. Ellers er det andre risikoer som har forskjellig alvorlighetsgrad.

Konsekvens	Risikovurdering				
5 - Veldig Alvorlig					
4 - Alvorlig		**2B, 3A**	**4A**		
3 - Moderat		**1A, 2C**	**1B, 5B**		
2 - Liten		**3B**	**2B, 5A**	**2A**	
1 - Veldig Liten					
Sannsynlighet	1 - Veldig Liten	2 - Liten	3 - Moderat	4 - Alvorlig	5 - Veldig Alvorlig

Ansvarlig person for aktivitet:	Jørgen Meland Lund									
Deltakere:	Jørgen Meland Lund, Jesper Vos, Henning Sønderland									
Forklaring av aktivitet:										
Bachelor Oppgave 3DOF										
Aktivitet	Mulige uønskede situasjoner	Eksisterende risikomottiltak	Vurdering av mulighet (S)	Evaluering av konsekvens (K)				Risiko (S x K)	Forslag til risikomottiltak	Risiko etter mottiltak (S x K)
			(1-5)	Menneskelig (1-5)	Material (1-5)	Ytre miljø (1-5)	Rykte (1-5)			
1. Drift av utstyr	1A. Kutt og blåmerker fra bruk av sag, bor, vinkelsliper og CNC-maskin	Sikkerhetstiltak bygd inn i utstyret	2	3	1	1	1	6	Gå gjennom sikkerhetstiltak og manualer før man bruker utstyret, og bruk nødvendig sikkerhetsutstyr	2
	1B. Metallspon og støv kommer i kontakt med øynene	Ingen	3	3	1	1	1	9	Bruk alltid vernebriller når man bruker utstyr	3
2. Arbeid i Manulab og Tunglab	2A. Snuble i rot og fall	Ingen	4	2	1	1	1	8	Hold omgivelsene våre rene og organiserte ved arbeid	2
	2B. Brann	Brannsluktingsapparat og sikkerhetstiltak	2	4	4	1	1	8	Sjekk regelmessig for brannfare og hold et brannslukningsapparat i nærheten	2
	2C. Innånding av farlige gasser	Ventilasjon	2	3	1	1	1	6	Sørg for å alltid ha god og skikkelig ventilasjon	3
3. Klargjøring av 3D-print og andre deler	3A. Farlige stoffer kommer i kontakt med øynene	Ingen	2	4	1	1	1	8	Hold hender og arbeidsstasjoner rene. Bruk vernebriller ved behov	2
	3B. Brannsår fra 3D-printerdysen	Ingen	2	2	1	1	1	4	Sørg for at 3D-printeren er av før rengjøring, og ikke kom for nær dysen	2
4. Betjening av laserkutteren	4A. Ser direkte på laseren i laserkutteren	Ingen	3	4	1	1	1	12	Dekk til øynene og pass på at man ikke ser direkte på laserkutteren	4
5. Avvik	5A. Sykdom blant gruppemedlemmer	Renholds rutiner	3	2	1	1	1	6	Holde seg hjemme ved mistanke om sykdom. Vær ekstra oppmerksom på kontakt med andre.	2
	5B. Dårlig framdrift/Ikke ferdig med oppgaver innenforgitt tidsramme	Ingen	3	1	1	1	3	9	Sørge for god oppfølging og eventuelt ekstra arbeid ved dårlig framdrift	3

5.6 Hovedaktiviteter i videre arbeid

- Beskrivelser av planlagte hovedaktiviteter og viktigste delaktiviteter for gjennomføring av prosjektet.

Nr	Hovedaktivitet	Ansvar	Kostnad i Kr	Tid/omfang
A1	Bygge stabil Platform	Alle	Maks 10K	Frist 28.02
A11	Laserkutte plate	Jesper	N/A	1 dag
A12	Bygge søyler	Jørgen	1K	1 uke
A13	Planlegge og bestille motorer	Henning	4K	2 uker
A2	Elektronikk	Alle	6K	3 uker
A21	Bestille og sette opp kamera	Alle	2K	2 uker
A22	Bli kjent med Khadas VIM	Alle	N/A	2 uker
A23	Koble sammen utstyr	Alle	N/A	1 uke
A3	Programmere	Alle	N/A	1 måned
A31	Koordinat innsamling av kamera	Alle	N/A	1 uke
A32	MPC	Alle	N/A	3 uker
Sum			16K	3 måneder

… med summering av planlagt ressursbehov og beregnet eller gitt tidsramme og økonomisk ramme.

5.7 Framdriftsplan – styring av prosjektet

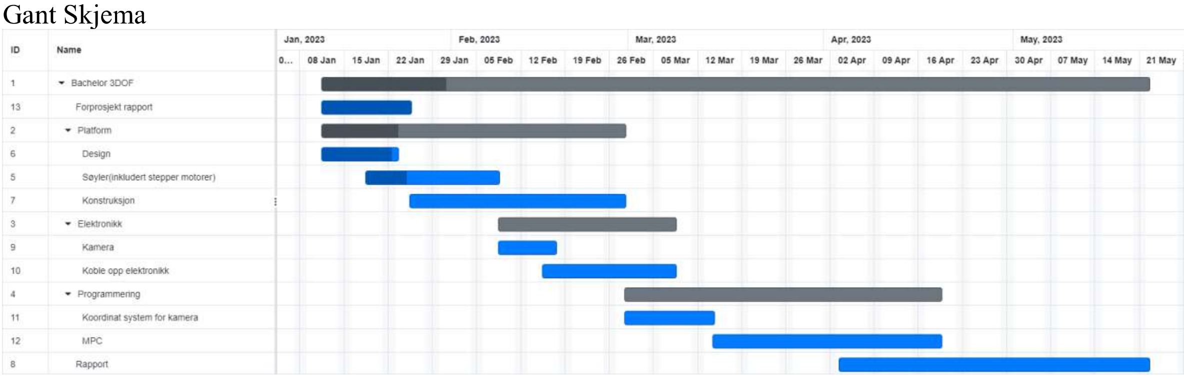
5.7.1 Hovedplan

Første oppgave er å komme fram til et design og konstruere platformen. Hovedsakelig vil dette bestå av tre deler. Disse delene er selve platen til platformen, Søylene og lagrene som skal brukes til å holde platformen oppe og sørge for stabil bevegelse, og fundamentet sammen med stepper motorene som skal brukes. Ansvaret for disse oppgavene er fordelt slik som i punkt 5.6.

Neste del er å bestille og koble sammen alt av elektronikk som trengs for å styre platformen. Dette vil bestå av kamera, eventuelle motor drivere, eventuelle enkodere, Khadas VIM mikrokontroller, og kabler. Det er enda ikke komt en felles beslutning for hovedplan og ansvar her enda.

Del 3 er å programmere kontroll av ballen. Dette skal gjennomføres ved å implementere MPC og koordinat samling av ballen i C++ eller Python. Dette skal hovedsakelig gjennomføres felles mellom alle medlemmer. Hovedansvaret her er enda ikke besluttet.

Gant Skjema



5.7.2 Styringshjelpemidler

Gantt-diagrammet vil være et styringshjelpemiddel når gruppen skal evaluere tidsbruk og planlegge for videre arbeid med prosjektet. Denne forrapporten vil også være en god kilde på hvordan den originale planen var satt opp, og gruppen kan dermed se tilbake på hva som skulle gjøres og hvor lang tid det skulle ta. En arbeidslogg vil også etableres for å dokumentere hvor lang tid som ble brukt og hvordan de ble løst. Kostnaden til de ulike komponentene vil bli sammenlignet med budsjettet som er satt opp i denne rapporten for å finne eventuelle avvik.

5.7.3 Utviklingshjelpemidler

Prosjektveilederen vil være det fremste hjelpemiddelet fordi han har erfaring som er relevant for prosjektet. Ellers er det mulig å finne informasjon på internett som kan bidra med å forstå hvordan ulike aspekt ved prosjektet skal løses. I tillegg til dette gruppen praktisere et flytskjema for å vise kodestrukturen.

5.7.4 Intern kontroll – evaluering

Evaluering gjennomføres av prosjektleder som følger opp framdrift. Kriterier for at et mål er oppnådd er at alle i gruppe er fornøyd med resultatet av den gitte aktiviteten.

5.8 Beslutninger – beslutningsprosess

Viktige beslutninger blir tatt ved diskusjon mellom alle gruppemedlemmene, og med prosjektveileder. Dette kan enten blir gjort på de planlagte møtene eller via mail dersom det skulle være tidskritisk. Oppgaven er ikke 100% definert, så det kommer til å bli diskusjoner underveis angående de tekniske detaljene i hvordan gruppen skal løse oppgaven.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Dokumentasjon som skal utarbeides:
CAD-filer
Python-kode
C-kode
Matlab-beregninger
Khadas (ubuntu/linux) oppsett
Gantt-skjema
Logg

Dokumentasjonen skal lagres på:
Lokale private PC'er
Onedrive
Teams
Overleaf
Felles fusion prosjekt mappe for 3D filer og design

Rutiner:
Regelmessige møter med prosjektveileder
Gode rutiner for lagring og versjonslogging
Huske å skrive arbeidslogg og tidsbruk

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

Etter samtale med prosjektveileder vil det bli møte annenhver onsdag for å diskutere fremdrift og få eventuell hjelp til problemløsning. Vi vil gi en liten rapport på hva so ble gjort siden sist møte, og hva som skal skje framover mot neste møte. Ellers vil vi alltid ha en god dialog på mail med veileder.

7.1.2 Prosjektmøter

Gruppen gjennomfører møter hver uke via Discord eller fysisk før arbeid starter. I disse møtene skal arbeidsoppgaver og eventuelle endringer diskuteres og fastsettes før arbeid startes.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)

Planlagte rapportformer
-Gant diagram
-Arbeidslogg

Planlagte rapportdatoer
– Annenhver uke i møte med veiledere

Framdriftsrapporter gjennomføres med Gant diagram og oppdateres annenhver uke før møte med veiledere.

8 PLANLAGT AVVIKSBEHANDLING

Dersom fremdriften ikke går som planlagt skal alle i gruppen møte fysisk og gjennomføre ekstra arbeid utover vanlig arbeidstider, slik at fremdriften ligger etter planen igjen. Ved endringer skal disse diskuteres først felles i gruppen, og deretter skal veiledere informeres om dette. Ansvaret for å følge med fremdriften og oppmøte ligger hos Jørgen og Jesper.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

Spesielt utstyr
Spesielt utstyr vi bli bestilt inn mest mulig rimelige priser etter avtale med prosjektveileder. Ellers vil gruppen bruke det utstyret og ressursene som er tilgjengelig på campus. NTNU sitt campus i Ålesund har lab-er med materialer og maskiner som dekker det meste av behovet til dette prosjektet.

9.1 Programvare

- PyCharm Professional edition Mulighet for ekstern koding
- CLion C++ IDE
- Matlab

9.2 Utstyr

- CNC maskin
- Laserkutter
- Loddebolt
- Varmluftspistol
- Skruer, bolter, muttere
- Induksjonsvarmer for kulelager
- 3D-Printer

10 REFERANSER

VEDLEGG

D.2 Report 08.02

Gruppe 2 Hovedprosjekt	Project 3DOF Platform med MPC	Antall møter denne perioden 1 planlagt	Firma - Oppdragsgiver NTNU Ålesund	Side 1 av 1
Framdriftsrapport	Period/week(s) 2	Antall brukte timer I henhold til logg 100	Prosjektgruppe (navn) Jesper Vos, Henning Sønderland, Jørgen Meland Lund	Dato 08.02.23

Hovudmål for denne perioden
- Sjekket at alt passer ilag ved å assemble alt og lage en modell i fusion 360.
- Få de fleste deler på plass

Planlagte aktiviteter for perioden
- Bygge steppermotor mounts
- 3D-printe deler
- Bestille kulelager
- Laserkutte plate
- Bestille steppermotorer

Faktisk gjennomførte aktiviteter denne perioden
- Laserkuttet plate
- Bestilt de fleste deler
- Designet platform i Fusion 360
- Assemblet modell i Fusion 360
- 3D-printet deler

Beskrivelse av hva som avviket fra aktivitetene og hvorfor
- Ikke bestilt motorer

Erfaringer fra denne perioden
- Vanskelig å finne passende motorer

Hovudfokus neste periode
- Bygge ferdig platformen
- Få i gang Khadas og kameraet

Planlagte aktiviteter neste periode
- Starte å teste kameraet, hvordan er FOV'en, kan man se gjennom platen?
- Finne en passende strømforsyning
- Se på HSV RGB colorspace, Cielab colorspace
- Oppdatere wikien
- Vurdere å starte å bruke github
- Kjøpe skjerm

Annet

Ønsker og bistand fra veileder
- Ikke noe spesielt

Signatur fra gruppemedlemmene	J.T, H.S, J.M.L

D.3 Report 23.02

Gruppe 2 Hovedprosjekt	**Project** 3DOF Platform med MPC	**Antall møter denne perioden** 1 planlagt	**Firma - Oppdragsgiver** NTNU Ålesund	Side 1 av 1
Framdriftsrapport	Period/week(s) 2	**Antall brukte timer I henhold til logg** 70	**Prosjektgruppe (navn)** Jesper Vos, Henning Sønderland, Jørgen Meland Lund	Dato 23.02.23

Hovudmål for denne perioden

- Bygge ferdig platformen
- Få i gang Khadas og kameraet

Planlagte aktiviteter for perioden

- Starte å teste kameraet, hvordan er FOV'en, kan man se gjennom platen?
- Finne en passende strømforsyning
- Se på HSV RGB colorspace, Cielab colorspace
- Oppdatere wikien
- Vurdere å starte å bruke github
- Kjøpe skjerm

Faktisk gjennomførte aktiviteter denne perioden

- Bygd ferdig plattformen
- Startet Khadas, og lagt inn software
- Startet testing av step-motorene
- Startet testing av kamera, prøvd med forskjellige linser (litt utfordring med FOV)
- Modifisert noen 3D deler for økt robusthet og styrke
-

Beskrivelse av hva som avviket fra aktivitetene og hvorfor

- Ikke kjøpt skjerm enda, ettersom vi ikke trenger den enda

Erfaringer fra denne perioden

- Lærte at det ikke ble så lett å finne kamera/linse som passer vårt bruk
- Tidskrevenede å laste software inn på Khadas

Hovudfokus neste periode

- Matematisk modell
- Finne et passende kamera/linse

Planlagte aktiviteter neste periode

- Styre steppermotorene med Khadas
- Lage et kordinatsystem for kameraet
- Koble alt det elektriske skikkelig

Annet

Ønsker og bistand fra veileder

- Hjelp til å sette opp Khadas skikkelig

Signatur fra gruppemedlemmene	J.T, H.S, J.M.L

D.4 Report 21.03

Gruppe 2 Hovedprosjekt	Project 3DOF Platform med MPC	Antall møter denne perioden 1 planlagt	Firma - Oppdragsgiver NTNU Ålesund	Side 1 av 2
Framdriftsrapport	Period/week(s) 2	Antall brukte timer I henhold til logg 100	Prosjektgruppe (navn) Jesper Vos, Henning Sønderland, Jørgen Meland Lund	Dato 21.03.23

Hovudmål for denne perioden
- Bygge ferdig platformen
- Få ut kordinater fra kamera
- Styre steppermotorene med høg presisjon

Planlagte aktiviteter for perioden
- Kode for å få ut kordinater fra kameraet
- Kjøre et State Space program for å teste
- Få presise målinger fra encoder
- Sette opp MPC
- Få til kommunikasjon mellom de forskjellige programma
- Forberede pitch presentasjon
- Oppdatere wikien

Faktisk gjennomførte aktiviteter denne perioden
- MPC er 90% ferdig
- Kjørt steppermotorene Arduino.
- Lest av alle tre encoder samtidig på arduino.
- Kamera koden gir oss presise kordinater
- Platformen ble designet for lett montering, samt for å være kompakt
- Bygd platformen

Beskrivelse av hva som avviket fra aktivitetene og hvorfor
- Ikke fått kjørt State Space program for å teste motorene, pga dårlig presisjon i motorene uten skikkelig feedback fra encoder.

Erfaringer fra denne perioden
- At MPC er omfattende
- At Khadas er dårlig på interupts og dermed upresis når man skal lese av encoder

Hovudfokus neste periode
- Teste State Space på platformen
- Få til MPC på platformen
- Lage GUI til skjermen

Planlagte aktiviteter neste periode
- Få til robust feedback fra encodere og styre motorene presist
- Programere GUI
- Ferdigstille MPC
- Teste med både State Space og MPC
- Bygge alt ferdig og koble alt fint opp

Annet

Ønsker og bistand fra veileder
- MPC generator bibliotek som Erlend nevnte på starten av året

Signatur fra gruppemedlemmene	J.T, H.S, J.M.L

Gruppe 2 Hovedprosjekt	**Project** 3DOF Platform med MPC	**Antall møter denne perioden** 1 planlagt	**Firma - Oppdragsgiver** NTNU Ålesund	Side 2 av 2
Framdriftsrapport	Period/week(s) 2	**Antall brukte timer I henhold til logg** 100	**Prosjektgruppe (navn)** Jesper Vos, Henning Sønderland, Jørgen Meland Lund	Dato 21.03.23

D.5 Report 11.04

Gruppe 2 Hovedprosjekt	**Project** 3DOF Platform med MPC	**Antall møter denne perioden** 1 planlagt	**Firma - Oppdragsgiver** NTNU Ålesund	Side 1 av 1
Framdriftsrapport	Period/week(s) 2	**Antall brukte timer I henhold til logg** 120	**Prosjektgruppe (navn)** Jesper Vos, Henning Sønderland, Jørgen Meland Lund	Dato 12.04.23

Hovudmål for denne perioden
- Kode ferdig og teste
- Koble ferdig elektronikken
- Komme i gang med rapporten

Planlagte aktiviteter for perioden
- Bruke multiprocessing og multithreading for å samkjøre koden
- Få instalert alle komponenter og koble opp alt
- Designe GUI til touchskjerm
- Skrive 10 sider på rapporten

Faktisk gjennomførte aktiviteter denne perioden
- 90% Ferdig kode
- 90% Ferdig bygging og oppkobling
- En funksjonibel GUI
- Skrevet 10 sider på rapporten

Beskrivelse av hva som avviket fra aktivitetene og hvorfor
- Encoder-kabler ble levert sent, og derfor ikke koblet opp enda
- Utfordringer med koden, og dermed ikke helt ferdig

Erfaringer fra denne perioden
- Bestille deler i tider
- Omfattende å samkjøre flere koder og opprettholde høg hastighet

Hovudfokus neste periode
- Bli ferdig, teste, kalibrere og optimalisere
- Gjøre mye på rapporten, for å kunne få feedback og veiledning

Planlagte aktiviteter neste periode
- Kjøre platformen med MPC, PID, og State Space
- Ferdigstille GUI
- Arbeide med rapporten

Annet

Ønsker og bistand fra veileder
- Feedback på rapporten, når vi har en betydelig mengde stoff å vise fram.

Signatur fra gruppemedlemmene	J.T, H.S, J.M.L

Appendix E

Video Links

Demonstration - https://www.youtube.com/watch?v=brYy_x_78rQ&t

Presentation - https://www.youtube.com/watch?v=V2SLBKUJTeg

NTNU

Norwegian University of
Science and Technology