

# VISUAL POSE ESTIMATION AND INFRASTRUCTURE DEVELOPMENT FOR DRONE REINFORCEMENT LEARNING ARENA

THOMAS LØNNE STIANSEN  
JAN LAKSESVELA HAUGSTAD  
ADRIAN MATHIAS LERVIK LING



SUPERVISOR  
Kristian Muri Knausgård

**University of Agder, 2024**  
Faculty of Engineering and Science  
Department of Engineering Sciences

# Preface

This bachelor's thesis is the final requirement for the mechatronics bachelor program, and marks the end of the three years with education at University of Agder. Despite a demanding workload, both the program and the project have been very exciting to work with and has given the students invaluable knowledge relevant to a engineering degree in mechatronics. The faculty's provision of technology such as the CNC-machines at the campus for rapid prototyping and the motion capture equipment has also enhanced the groups motivation for the project.

This project would not have been possible without the help from our proficient supervisor Kristian Muri Knausgård, whom we want to say thank you for the guidance and accessibility throughout the semester. We would also like to thank the rest of the faculty and its excellent staff, which have been most accommodating with both theoretical and practical assistance during both the project and the rest of the degree.

Thank you all for a great collaboration.

Adrian Ling  
Adrian Mathias Lervik Ling

Thomas L. Stiansen  
Thomas Lønne Stiansen

Jan Laksesvæla Haugstad  
Jan Laksesvæla Haugstad

Grimstad, May 14, 2024

A video of the development process and the system in action is available at:

<https://youtu.be/jgXzDRpN-2Y>

The full repository from version control of the project is available at:

[https://github.com/AMLing2/MAS306\\_Drone\\_G12](https://github.com/AMLing2/MAS306_Drone_G12)

# Abstract

This thesis incorporates the design process of a machine vision system for estimating the pose of a quadcopter inside the drone reinforcement learning arena from the 2022 UiA mechatronics BSc project [14]. The visual pose estimation is compared to a reference characteristic through the use of high quality performance motion capture equipment. This was made reproducible by developing a test with customized tools, test plan, scripts and more. Further analysis of the data from the tests was done to demonstrate the potential of the visual system and its possible role for providing low rate measurements to a state estimator in the future of the arena. A simplified filter algorithm with position and velocity is simulated with kinematics for translational motion to show how part of the vision system could be implemented in a future state estimator. The thesis also contains development of the system's infrastructure, entailing framework, communication and the connections between the arena's modules like the stepper and servo motors, as well as a redesign of the PCB for the actuator of the arena. Finally, the project also includes the design and development of a new airframe for the drone; ensuring proper protection of the components which should endure the many iterations of the flight controlled by reinforcement learning. The mechanical drone design is supported by impact simulations and computational fluid dynamics (CFD) simulations.

# Nomenclature

## Abbreviations and Acronyms

COG	Center of Gravity
COG	Computational Fluid Dynamics
DOF	Degrees of Freedom
EKF	Extended Kalman Filter
FC	Flight Controller
FHD	Full High-Definition
FOV	Field of View
FPS	Frames Per Second
GG	Global Goal
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HD	High-Definition
HMI	Human-Machine Interface
I2C, I <sup>2</sup> C	Inter-Integrated Circuit
IDE	Integrated Development Environment
INS	Inertial Navigation Systems
IO	Input/Output
IR	Infrared
LTS	Long Term Support
MCU	Microcontroller Unit
MOSFET	Metal Oxide Semiconductor Field-Effect Transistor
NED	North-East-Down (coordinate frame)
PDF	Probability Distribution Function
PF	Particle Filter
PP	Polypropylene
QTM	Qualisys Track Manager
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
SDK	Software Development Kit

SG	Surface Goal
SLAM	Simultaneous Localization and Mapping
SPI	Serial Peripheral Interface
SPS	Samples Per Second
SSE	Sum of Square Error
SSR	Sum of Squares of Regression
SST	Total Sum of Squares
UAV	Unmanned Aerial Vehicle
UKF	Unscented Kalman Filter
UUT	Unit Under Test

### Common Mathematical Notations

$(c_x, c_y)$	Principal Point (Horizontal, Vertical)
$(f_x, f_y)$	Focal Length (Horizontal, Vertical)
$\mathcal{N}$	Normal Distribution
$\mu$	Average/Mean/Expected Value
$\phi$	Angle about the x axis: Roll
$\psi$	Angle about the z axis: Yaw
$\sigma$	Standard Deviation
$\sigma^2$	Variance
$\theta$	Angle about the y axis: Pitch
$v_k$	Measurement Noise
$w_k$	Process Noise

# Contents

<b>Preface</b>	i
<b>Abstract</b>	ii
<b>Nomenclature</b>	iv
<b>1 Introduction</b>	1
1.1 Background and Motivation . . . . .	1
1.2 State of Art . . . . .	2
1.3 Problem Statement . . . . .	2
1.4 Report Structure . . . . .	3
<b>2 Theory</b>	5
2.1 Camera History and The Pinhole Model . . . . .	5
2.2 Coordinate Frame Transformations in $R^3$ . . . . .	6
2.2.1 Vector Translation . . . . .	7
2.2.2 Rotation Formalisms . . . . .	7
2.2.2.1 Rotation Matrices . . . . .	7
2.2.2.2 Euler Angles . . . . .	8
2.2.2.3 Rotation Vector (Euler Axis and Angle) . . . . .	9
2.2.2.4 Quaternions . . . . .	9
2.3 Fiducial Markers . . . . .	10
2.4 Machine Vision . . . . .	11
2.4.1 Intrinsic Parameters . . . . .	11
2.4.2 Camera Geometry . . . . .	12
2.4.2.1 Homogenous Coordinates and Extrinsic/Intrinsic Camera Matrices .	12
2.4.2.2 Camera Calibration: Zhang's Method . . . . .	14
2.4.3 Computer Vision Library: OpenCV . . . . .	16
2.4.3.1 OpenCV: Camera Calibration . . . . .	16
2.4.3.2 OpenCV: Pose Estimation . . . . .	17
2.5 Intel RealSense D435 . . . . .	18
2.5.1 Depth Module . . . . .	18
2.5.2 RGB Camera . . . . .	19
2.5.3 Infrared Projector . . . . .	19
2.5.4 Vision Processor D4 . . . . .	20
2.6 Wrappers . . . . .	20
2.7 Statistics . . . . .	20
2.7.1 Normal Distribution . . . . .	21
2.8 State Space Representation . . . . .	22
2.9 Kalman Filter . . . . .	23
2.10 Software Development . . . . .	24
2.10.1 Object oriented programming and polymorphism . . . . .	24
2.10.2 Threading and parallelism . . . . .	25
2.10.3 PC timers . . . . .	25
2.10.4 Registers and bitwise operations . . . . .	26
2.11 Communication within the drone arena system . . . . .	26
2.11.1 UDP and socket programming . . . . .	26
2.11.2 Inter-process communication . . . . .	27

2.11.3	Data packaging . . . . .	27
2.11.4	SPI on Raspberry PI . . . . .	27
2.11.5	Buildroot operating system . . . . .	28
2.12	Lead screw transmission . . . . .	28
2.13	MOSFET . . . . .	29
2.14	Circuit protection and Decoupling capacitors . . . . .	29
2.15	Trace width . . . . .	30
2.16	Inertial Measurement Unit (IMU) . . . . .	30
2.17	Fundamental Aerodynamics Relevant to The Quadcopter . . . . .	31
2.18	Drone Pendulum Fallacy . . . . .	32
<b>3</b>	<b>Concept Development</b>	<b>33</b>
3.1	Overview and requirements . . . . .	33
3.2	Concept Ideas . . . . .	33
3.2.1	Concept 1: Arena Centered . . . . .	34
3.2.2	Concept 2: Drone Centered . . . . .	34
3.2.3	Concept 3: Hybrid Solution . . . . .	34
3.3	Concept Evaluation . . . . .	35
<b>4</b>	<b>Design and implementation</b>	<b>37</b>
4.1	System Architecture and Design . . . . .	37
4.2	Project Workflow . . . . .	38
4.2.1	Version Control . . . . .	38
4.2.2	Project Management . . . . .	38
4.3	Linux Computer Setup . . . . .	38
4.3.1	Desktop Setup . . . . .	38
4.3.2	Secure Shell and X11 Forwarding . . . . .	39
4.3.3	Arena Raspberry PI setup . . . . .	40
4.4	Intel Realsense D435 Setup . . . . .	40
4.4.1	Camera Information - Software Development Kit Setup . . . . .	41
4.4.2	Camera Readings . . . . .	41
4.5	Machine Vision Development - Outside of Arena . . . . .	42
4.5.1	Marker Detection . . . . .	42
4.5.2	Intrinsic Parameters . . . . .	43
4.5.3	Marker Pose Estimate . . . . .	43
4.5.4	Camera Calibration and Translation Measurement Test . . . . .	43
4.5.5	Depth Readings . . . . .	44
4.5.6	Combined Depth and Marker Estimate . . . . .	44
4.6	Machine Vision Development - inside the Arena . . . . .	45
4.6.1	Marker Dictionary Test . . . . .	45
4.6.2	Revised Machine Vision Algorithm . . . . .	45
4.7	Pose Validation with Qualisys . . . . .	47
4.7.1	Pose Validation Test Concept . . . . .	47
4.7.2	Pose Validation Test Development . . . . .	47
4.7.2.1	Physical Development . . . . .	47
4.7.2.2	Software Development . . . . .	48
4.7.3	Plotting . . . . .	50
4.7.4	Translation Distribution Fitting . . . . .	51
4.7.5	Position Kalman Filter: 1 DOF . . . . .	52
4.7.6	Position Kalman Filter: 3 DOF . . . . .	54
4.8	Development of a server for the drone arena . . . . .	55
4.8.1	Concept . . . . .	55
4.8.2	Working method of the server system . . . . .	56
4.8.2.1	Server . . . . .	56

4.8.2.2	Client packages . . . . .	58
4.8.3	Creating the server system . . . . .	58
4.8.3.1	Protocol Buffer Messages . . . . .	58
4.8.3.2	Synchronisation method . . . . .	59
4.8.3.3	Server . . . . .	60
4.8.3.4	Client - C++ . . . . .	62
4.8.3.5	Client - Python . . . . .	62
4.8.4	Using CMake and GDB . . . . .	63
4.9	Arena actuator improvements . . . . .	64
4.9.1	Design process . . . . .	64
4.9.1.1	Azimuth sensor . . . . .	64
4.9.1.2	Endstop switch . . . . .	65
4.9.1.3	Communication with the actuator system and the server . . . . .	66
4.9.2	Actuator PCB . . . . .	67
4.9.2.1	Soldering the PCB . . . . .	69
4.10	Drone Hardware and Software . . . . .	71
4.10.1	Buildroot . . . . .	71
4.10.2	IMU . . . . .	71
4.10.2.1	Troubleshooting the IMU SPI using a logic analyzer . . . . .	71
4.10.2.2	Reading and calibrating the IMU . . . . .	76
4.11	Mechanical Drone Design . . . . .	77
4.11.1	Material Choice and Polypropylene 3D-printing . . . . .	77
4.11.2	Geometric design . . . . .	77
4.11.3	Computational Fluid Dynamics (CFD) Simulation . . . . .	78
4.11.4	Impact Test Simulations . . . . .	79
4.11.5	Physical Tests . . . . .	79
<b>5</b>	<b>Results</b> . . . . .	<b>81</b>
5.1	Intel RealSense D435 . . . . .	81
5.1.1	Camera Information . . . . .	81
5.1.2	Camera readings . . . . .	83
5.2	Machine Vision . . . . .	85
5.2.1	Marker Detection . . . . .	85
5.2.2	Intrinsic Parameters . . . . .	85
5.2.3	Camera Calibration . . . . .	86
5.2.4	Manual Measurement Test . . . . .	88
5.2.5	Depth Readings . . . . .	89
5.2.6	Combined Marker Pose Estimate and Depth Reading . . . . .	90
5.2.7	Marker Dictionary Test . . . . .	90
5.3	Pose Validation with Qualisys . . . . .	91
5.3.1	Physical Problems and Solutions . . . . .	91
5.3.2	Calibration and Setup . . . . .	91
5.3.3	Plotting Results . . . . .	94
5.3.3.1	Test #2 Results . . . . .	95
5.3.3.2	Test #3 Results . . . . .	99
5.3.4	Translation Distribution Fitting . . . . .	103
5.3.5	Position Kalman Filter: 1 DOF . . . . .	106
5.3.6	Position Kalman Filter: 3 DOF . . . . .	108
5.4	Server . . . . .	112
5.4.1	Message transfer test . . . . .	112
5.4.2	Delay test . . . . .	112
5.5	Actuator . . . . .	113
5.5.1	USB communication . . . . .	113
5.5.2	Endstop Button . . . . .	113

5.5.3	Encoder . . . . .	113
5.6	Drone Hardware and Software . . . . .	115
5.6.1	IMU Readings . . . . .	115
5.6.1.1	Accelerometer gravity readings . . . . .	115
5.6.1.2	Gyroscope movement readings . . . . .	115
5.6.2	Buildroot . . . . .	116
5.7	Mechanical Drone Design . . . . .	116
5.7.1	Geometric Design . . . . .	116
5.7.2	3D Printing . . . . .	118
5.7.3	Computational Fluid Dynamics Simulations . . . . .	120
5.7.4	Impact test simulations . . . . .	122
5.7.4.1	”Flat Fall”: 1.5 meter fall landing square onto arena floor . . . . .	122
5.7.4.2	”Vertical Fall”: 1.5 meter fall with 90° rotation - 1 propeller guard impact . . . . .	122
5.7.4.3	”Tilted Fall 1”: 1.5 meter fall with 45° rotation - 1 propeller guard impact . . . . .	123
5.7.4.4	”Tilted Fall 2”: 1.5 meter fall with 45° rotation - 2 propeller guards impact . . . . .	123
5.7.4.5	Verifying stress concentration . . . . .	124
5.7.5	Physical Tests . . . . .	124
<b>6</b>	<b>Discussion</b> . . . . .	<b>125</b>
6.1	System Architecture and Design . . . . .	125
6.2	Intel RealSense D435 Setup . . . . .	125
6.2.1	Camera Information . . . . .	125
6.2.2	Camera Readings and Configurations . . . . .	126
6.2.2.1	Color Stream . . . . .	126
6.2.2.2	Infrared Stream . . . . .	126
6.2.2.3	Depth Stream . . . . .	126
6.2.2.4	Final Configurations . . . . .	127
6.3	Machine Vision Development . . . . .	127
6.3.1	Marker Detection . . . . .	127
6.3.2	Intrinsic Parameters . . . . .	127
6.3.3	Marker Pose Estimate . . . . .	127
6.3.4	Camera Calibration . . . . .	127
6.3.5	Manual Measurement Test . . . . .	127
6.3.6	Depth Readings and Combined with Marker Pose Estimate . . . . .	128
6.3.7	Marker Dictionary Test . . . . .	128
6.3.8	Revised Machine Vision Algorithm . . . . .	128
6.4	Pose Validation Test . . . . .	129
6.4.1	Physical Problems and Solutions . . . . .	129
6.4.2	Calibration and Setup . . . . .	129
6.4.3	Plotting Results . . . . .	130
6.4.4	Translation Distribution Fitting . . . . .	131
6.4.5	Position Kalman Filter: 1 DOF . . . . .	131
6.4.6	Position Kalman Filter: 3 DOF . . . . .	132
6.4.7	Further Work: State Estimator . . . . .	132
6.5	Server . . . . .	133
6.5.1	Further development . . . . .	133
6.6	Actuator . . . . .	135
6.6.1	Absolute encoder . . . . .	135
6.6.2	Communication . . . . .	136
6.6.2.1	USB communication . . . . .	136
6.6.2.2	Server . . . . .	136

6.6.3	PCB design . . . . .	136
6.7	Drone Hardware and Software . . . . .	137
6.7.1	Buildroot OS . . . . .	137
6.7.2	IMU . . . . .	137
6.8	Mechanical Drone Design . . . . .	137
6.8.1	Physical Design Discussion . . . . .	137
6.8.2	Computational Fluid Dynamics Simulations . . . . .	138
6.8.3	Physical and Simulated Impact Tests . . . . .	139
<b>7</b>	<b>Conclusion</b>	<b>141</b>
<b>Appendices</b>		<b>A - 1</b>
<b>A Technical Drawings</b>		<b>A - 1</b>
A.1	Pose Validation Test . . . . .	A - 2
A.1.1	Marker Cube . . . . .	A - 2
A.1.2	Calibration Plate . . . . .	A - 3
A.1.3	Actuator Bracket . . . . .	A - 4
A.2	Arena Development . . . . .	A - 5
A.2.1	Magnet Holder . . . . .	A - 5
A.2.2	Button Block . . . . .	A - 6
A.2.3	Card Holder . . . . .	A - 7
A.3	Drone Design . . . . .	A - 8
A.3.1	Drone Assembly . . . . .	A - 8
A.3.2	Airframe Bottom . . . . .	A - 9
A.3.3	Airframe Top . . . . .	A - 10
A.3.4	Airframe Core . . . . .	A - 11
A.3.5	Battery Bracket . . . . .	A - 12
<b>B Bill of materials</b>		<b>B - 1</b>
B.1	Bill of materials PCB . . . . .	B - 1
<b>C Arena PCB Schematic</b>		<b>C - 1</b>
<b>D Project Management</b>		<b>D - 1</b>
D.1	Gantt Chart . . . . .	D - 1
D.2	Kanban Board . . . . .	D - 3
<b>E Pose Validation Test</b>		<b>E - 1</b>
E.1	Test Plan . . . . .	E - 1
E.2	File Overview . . . . .	E - 5
E.3	Pose Validation Test Results: Plots from Full Time Range . . . . .	E - 6
E.3.1	Test #0 Results . . . . .	E - 6
E.3.2	Test #1 Results . . . . .	E - 10
E.3.3	Test #2 Results . . . . .	E - 14
E.3.4	Test #3 Results . . . . .	E - 18
E.3.5	Test #4 Results . . . . .	E - 22
E.4	1D Position Kalman Filter Results . . . . .	E - 26
E.4.1	Test #0 . . . . .	E - 26
E.4.2	Test #1 . . . . .	E - 27
E.4.3	Test #2 . . . . .	E - 29
E.4.4	Test #3 . . . . .	E - 30
E.4.5	Test #4 . . . . .	E - 32
E.5	3D Position Kalman Filter Results . . . . .	E - 34
E.5.1	Test #0 . . . . .	E - 34

E.5.2	Test #1 . . . . .	E - 37
E.5.3	Test #2 . . . . .	E - 40
E.5.4	Test #3 . . . . .	E - 43
E.5.5	Test #4 . . . . .	E - 46
<b>F</b>	<b>Buildroot</b>	<b>F - 1</b>
F.1	Buildroot OS for RPi CM4 . . . . .	F - 1
F.2	Cross compiling for the CM4 and running test program . . . . .	F - 1
<b>G</b>	<b>Source Code</b>	<b>G - 2</b>
G.1	Machine Vision Development . . . . .	G - 2
G.1.1	depthCameraTest.py . . . . .	G - 2
G.1.2	screenshotCamera.py . . . . .	G - 3
G.1.3	arucoDetection.py . . . . .	G - 5
G.1.4	arucoPoseEstimation.py . . . . .	G - 7
G.1.5	cameraCalibration.py . . . . .	G - 10
G.1.6	depthCamera.py . . . . .	G - 12
G.1.7	poseMarkerDepthPreArena.py . . . . .	G - 14
G.1.8	recordingCamera.py . . . . .	G - 17
G.1.9	markerTestAnalyzer.py . . . . .	G - 18
G.1.10	markerTestVideoExport.py . . . . .	G - 20
G.1.11	computerVisionMain.py . . . . .	G - 22
G.2	Pose Validation Test . . . . .	G - 25
G.2.1	poseTestInitializer.py . . . . .	G - 25
G.2.2	cameraCenter.py . . . . .	G - 27
G.2.3	poseTestRecording.py . . . . .	G - 28
G.2.4	poseTestArucoExport.py . . . . .	G - 30
G.2.5	poseTestSynchExport.py . . . . .	G - 33
G.2.6	qualisysPlotting.m . . . . .	G - 37
G.2.7	distributionFitting.m . . . . .	G - 49
G.2.8	KalmanFilter1D.m . . . . .	G - 54
G.2.9	KalmanFilter3D.m . . . . .	G - 60
G.3	Server . . . . .	G - 69
G.3.1	dronePosVec.proto . . . . .	G - 69
G.3.2	Server Cmake . . . . .	G - 71
G.3.3	arenaServer.h . . . . .	G - 72
G.3.4	messengerClass.cpp . . . . .	G - 77
G.3.5	serverMain.cpp . . . . .	G - 87
G.3.6	msgThreads.cpp . . . . .	G - 90
G.4	Clients - C++ . . . . .	G - 92
G.4.1	drone cmake . . . . .	G - 92
G.4.2	arenaHeader.h . . . . .	G - 93
G.4.3	droneMain.cpp . . . . .	G - 96
G.4.4	clientimplementation.cpp . . . . .	G - 98
G.4.5	droneIMU.h . . . . .	G - 107
G.4.6	IMUimplementation.cpp . . . . .	G - 109
G.5	Clients - Python . . . . .	G - 114
G.5.1	arenaComm.py . . . . .	G - 114
G.5.2	RLsendRecvEx.py . . . . .	G - 121
G.6	Arena . . . . .	G - 122
G.6.1	bmi088.cpp . . . . .	G - 122
G.6.2	i2cAS5600.cpp . . . . .	G - 124
G.6.3	azionly.py . . . . .	G - 126
G.6.4	wiringpitest.cpp . . . . .	G - 128

G.7	Drone . . . . .	G - 130
G.7.1	IMU test reading . . . . .	G - 130
G.7.2	IMU only code . . . . .	G - 132
G.7.2.1	CmakeListst.txt . . . . .	G - 132
G.7.2.2	droneIMU.h . . . . .	G - 133
G.7.2.3	imuImplementation.cpp . . . . .	G - 135
G.7.2.4	imuMainTest.cpp . . . . .	G - 141

## Bibliography

**G - 143**

*This page intentionally left blank.*

# 1 Introduction

## 1.1 Background and Motivation

This thesis' roots can be traced back to 2017 with the preceding drone related projects. The project lineup has since evolved from the original "*3 DOF Quadcopter Platform*" [49], where the 2022 UiA mechatronics BSc. project took an AI-oriented turn with the "*Design and Development of a Drone Reinforcement Learning Arena*" [14]. This thesis incorporates further development on the arena and focuses on preparing for reinforcement learning with the design of a working infrastructure, a resilient airframe design for the drone and development of a visual pose estimation system which can be used with a state estimator to approximate the hidden state of the drone for better control. A popular approach from aided navigation is utilizing high rate sensors such as an IMU for acceleration and angular velocity measurements together with lower rate position measurements from GPS or GNSS [37]. Given the restricting volume of the arena, a vision system may be more viable than GPS/GNSS; additionally it may be replaced with a GPS module for future state estimators together with the rest of the designed system, if the drone projects expand to outside flight in the future. This combination of sensors is an approach where a state estimator presents information about the drone's state including position, velocity, angle and angular velocity; much more precise than sensors alone can provide. In addition to providing hidden information about the drones state, an increased bandwidth of the estimate's rate is a favored attribute of the estimator, as it can update based a subset of sensors and the mathematical model even when not all are available. This report will contain relevant mathematical representations for expressing the pose and certain states of the drone, as well as statistics for quantizing the performance for parts of the developed visual pose estimation. The project aims to provide the university with a physical solution in the future for both educational purposes and demonstration of RL at different stages of iterations. Some of the methods used in this thesis have previously been explored in the project lineup, therefore providing valuable insight for this years development.

## 1.2 State of Art

The drone projects at UiA have provided valuable information, especially for this project. Furthermore, there is a lot of external information relevant to this years project. Autonomous drones are being researched across the globe, ranging from automation in the industry to recreational purposes such as drone races. As such unmanned aerial vehicles (UAV)s are complex systems which may be subject to unpredictable environments and situations, various methods have been explored in order to achieve desired behaviour. Secondary research indicates that autonomous drone development is mostly done through simulations, like the master project from 2021 at UiA with the development of a state estimator with simulation using a hybrid Kalman filter and particle filter for indoor navigation [64]. State space representation is a known method to represent the various situations of systems, and is extensively documented and compatible with several filters in order to achieve the most correct estimate possible. Compatible with the state space representation are various state estimators, which are well known and documented. Inertial navigation systems (INS) applications are relevant to this project, with extensive research and theory can be found in books like "Aided Navigation" by Farrell [37], as well as the previous projects in the lineup. Physical approaches to autonomous drones are also becoming gradually more popular; a group of students from University of Zurich beat manually operated drones with their autonomous drone, achieving a speed exceeding 100 [km/h] [108]. The autonomous drone race where this group competed, is a testing arena which takes place in a hangar on former grounds of Dübendorf Air Base [69]. Other than this, physical arenas for autonomous drones are limited.

There are many other specific technologies related to this project, such as communication protocols and frameworks like ROS2. Central for the visual sensors is machine vision, with substantial growth the last years. It has become an increasingly vital part of robotics, with many tools facilitating the research and development process. OpenCV is a widely known library created in C++, with support for fiducial markers to more easily track objects in the image [81]. Measuring the performance of systems during development is important, and is streamlined with modern technology such as the motion capture equipment at the faculty.

## 1.3 Problem Statement

As the final product needs more knowledge of the reinforcement learning process, this years project will focus on preparing the drone RL arena for state estimation and the infrastructure of the arena system, along with improvements on the arena's actuators and the drone airframe. As mentioned in section 1.1, part the goal of this years project is to develop a vision system to provide information of the quadcopter's pose in order to give quantified data to the RL model. The system is planned to have real time requirements, but as far as the visual system goes this is not as strict since the state estimator can provide higher rate of the state estimate than the measurement rate. To use the visual pose measurements for a state estimator, values quantifying the performance is desired and therefore high priority in this project. The arena system lacks a method for communication which is required to combine the many running processes for the system to work completely. Therefore a framework for the communication needs to be implemented, where part of the scope for the communications is to quantify the real-time capabilities such as the delay. The actuator system of the arena which is used to flip, rotate, and charge the drone is also not complete for a full flight cycle, so the actuator system will also be developed further. As the previous drone design insufficiently protects the propellers, a new prop guard design is for preventing damage to both them and the rest of the drone. The new design should give the current group more flexibility for designing the visual system, as the airframe is to be redesigned to fit fiducial markers. The new design should also allow for a freer passage of propeller downwash for increased efficiency.

## **1.4 Report Structure**

Prior to the development process is chapter 3, which was made in the earliest stages of the semester for presenting the concepts for the general structure of the project. The rest of the report is mainly divided into three parts, which applies to chapters 2, 4, 5 and 6.

The first part is the development of the machine vision system and the corresponding validation test. As mentioned later in the report, prior to reading about the pose validation test in section 4.7, it is advised to go through the test plan in appendix E.1. The second part of the report is centered around the development of the framework and communication of the system, as well as improvements of the electrical parts of the physical arena. The third part is centered primarily around the mechanical drone design.

*This page intentionally left blank.*

# 2 Theory

## 2.1 Camera History and The Pinhole Model

Various cameras have different properties and can therefore be represented with several unique models, such as the Kannala-Brandt model for representing distortion in wide-angle, ultra wide-angle, and fish-eye lenses [109]. However, the basic principle behind cameras stay the same and can be illustrated with the pinhole model.

Before explaining the pinhole model, a brief review of the cameras history will be provided. The predecessor to the modern cameras as most know it, is the camera obscura which has been known since the 4th century BC [47]. The name "camera obscura" is latin for "dark room" [47], reflecting its working principle. It works by having light go through a small hole (pinhole) in a wall and hit the wall on the other side, which is inside a dark room. The camera obscura was the simplest demonstration of this, as it was just a projection which does not capture the image.

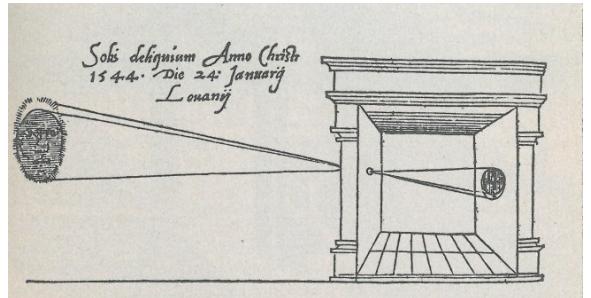


Figure 2.1: Earliest illustration of the camera obscura from Gemma Friscus' book "De Radio Astronomica et Geometrica", 1545 [111].

The projected image is inverted, due to how the light rays travel through the pinhole. An illuminated object sends out rays of light in all directions. The pinhole ensures that only one small portion of the light from each part of the object penetrates the wall. Tracing the rays from a set of the objects points towards and past the pinhole makes it apparent why the projected image is inverted. A larger pinhole opening would restrict less of the rays, which leads to a brighter image with rays overlapping. This leads to a blurred image as the rays interfere with each other. A too small pinhole may also lead to a blurred image, as the rays might refract due to a phenomenon called diffraction.

This pinhole effect has since been used for the development of photographic cameras, where the projected image is captured. Initially done through physical chemistry, later converted to the digital camera by capturing the image with sensors. The pinhole model is a simplified representation of these photographic cameras, based on the same principles as the camera obscura. It provides a representation of the cameras geometry, as well as a baseline for camera parameters. There are several versions of the pinhole model with slight alterations such as the orientation of the x and y axes for the image and focal plane. Another common difference between the models is the location of the image plane being in front of or behind the pinhole. The pinhole model used by OpenCV has the frontal plane version, which is shown in figure 2.2, where the image plane is in front of the camera center  $F_C$ . The distance between the image plane and focal plane is called the focal length  $f$ . A shorter focal length results in a wider field of view (FOV) and more distortion. A longer focal length has a more narrow FOV, but has less distortion. This is directly related to how the light rays travel through the pinhole. With this model, a point in the real world is projected onto the image plane with pixel coordinates. Realistically it goes together with all other light rays through the camera center before reaching the sensors on the other side of the focal plane, resulting in an inverted image.

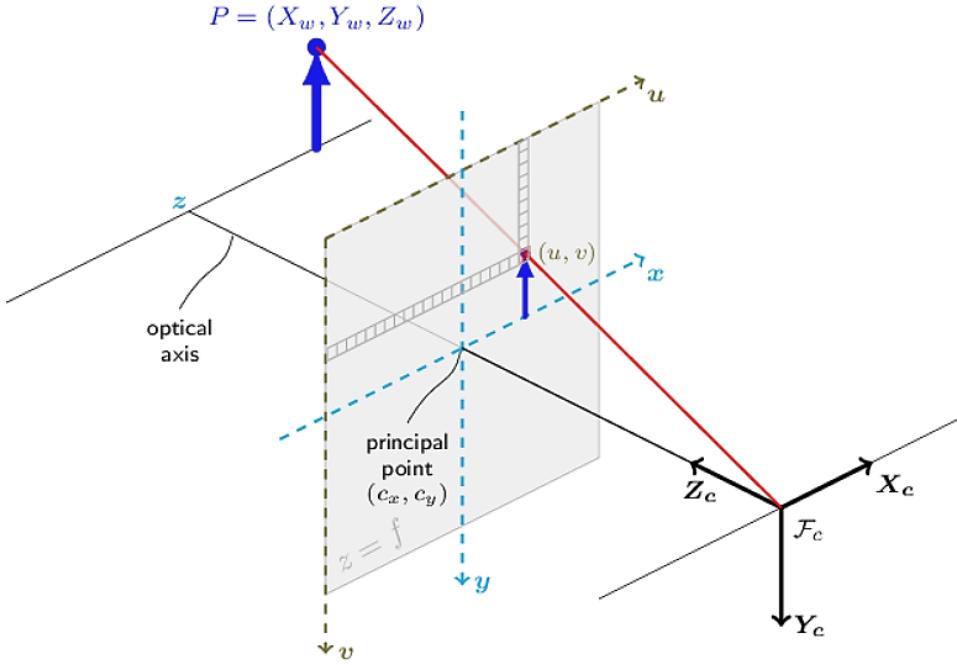


Figure 2.2: Pinhole Model used by OpenCV [83]

## 2.2 Coordinate Frame Transformations in $R^3$

There are two main properties when moving between coordinate frames (or reference frames) for representing an object in three dimensions, namely translation and orientation (also called attitude). The former refers to how one relates the various coordinate frames' origin locations, whilst the latter represents the direction of each axis. This is commonly done in aeronautics (amongst many other applications such as augmented reality), where the state of the object often requires multiple representations. Many applications benefit from multiple reference frames, as certain sensors may be mounted on moving systems and provide internal information without knowledge of the outside world. Furthermore, it enables the possibility to display the same state from any point of view. An example of relevant reference frames are shown in figure 2.3.

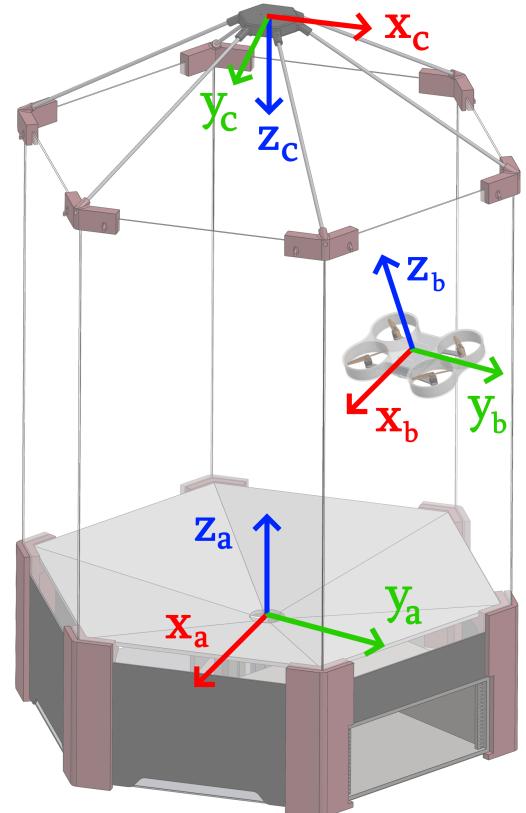


Figure 2.3: Relevant reference frames: camera, body and arena

### 2.2.1 Vector Translation

The translation between the different reference frames can be done with simple vector arithmetic. This is based on their center locations being represented as a vector, relating the center coordinates to the current frame. The translation between the frames have many applications where one desires to observe the position of the object relative to another reference frame, such as moving the quadcopter's position from the camera coordinate frame to the arena frame. This can be done by subtracting the coordinates of the initial point from the coordinates of the terminal point [7]:

$$\overrightarrow{AD} = \overrightarrow{CD} - \overrightarrow{CA} = \begin{bmatrix} X_{CD} - X_{CA} \\ Y_{CD} - Y_{CA} \\ Z_{CD} - Z_{CA} \end{bmatrix} \quad (2.1)$$

Where  $\overrightarrow{AD}$  is the coordinates of the drone from the arena reference frame,  $\overrightarrow{CD}$  is the coordinates of the drone from the camera reference frame and  $\overrightarrow{CA}$  is the coordinates of the arena from the camera reference frame.

### 2.2.2 Rotation Formalisms

Various rotation formalisms exist for expressing the orientation of a 3-dimensional object, with their own advantages and disadvantages. This report will primarily focus on four transformations, namely rotation matrices, Euler rotations, rotation vectors and quaternions. The basis for 3D rotation is a set of three unit vectors, representing the orthogonal axes of the object's local reference frame.

#### 2.2.2.1 Rotation Matrices

Rotation matrices provide a simple and efficient way to relate orientations of reference frames. The transformation is expressed as an orthogonal basis matrix with a determinant of 1 [132], which represents the transformation on the three axes where each column consists of the corresponding basis vector defining the reference frame:

$$A = [\hat{u} \mid \hat{v} \mid \hat{w}] = \begin{bmatrix} \hat{u}_x & \hat{w}_x & \hat{w}_x \\ \hat{u}_y & \hat{w}_y & \hat{w}_y \\ \hat{u}_z & \hat{w}_z & \hat{w}_z \end{bmatrix} \quad (2.2)$$

Before elaborating further on rotation matrices in  $R^3$ , consider a rotation of the standard basis vectors  $e_1$  and  $e_2$  on the x- and y-axis respectively, in  $R^2$ . It can be expressed as a matrix with columns consisting of trigonometric functions, mapping both vectors to a counterclockwise rotation about the origin with the angle  $\theta$ :

$$A = [T(e_1) \mid T(e_2)] = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (2.3)$$

For consistency, this report will maintain counterclockwise rotation as positive to comply with the "right hand rule".

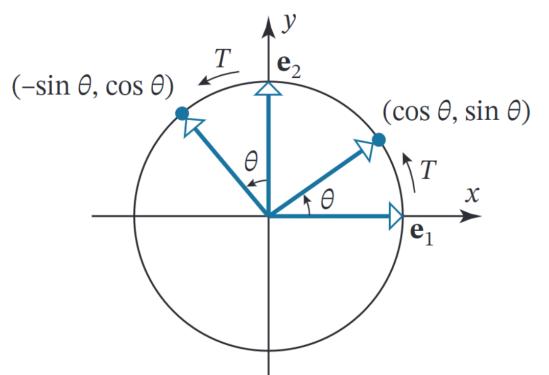


Figure 2.4: Two dimensional rotation [7]

The 3-dimensional transformation is simple to derive from equation (2.3), resulting in a transformation with matrix multiplication on the three orthogonal unit vectors representing the object's attitude. The three elementary rotations about the frame's axes are called roll, pitch and yaw, with the angle noted as  $\phi$ ,  $\theta$  and  $\psi$ . These are shown in figure 2.5, which also illustrates how the rotations build on each other, highlighting that the order matters as the transformation is not commutative (for instance z-y-x is not necessarily the same as z-x-y).

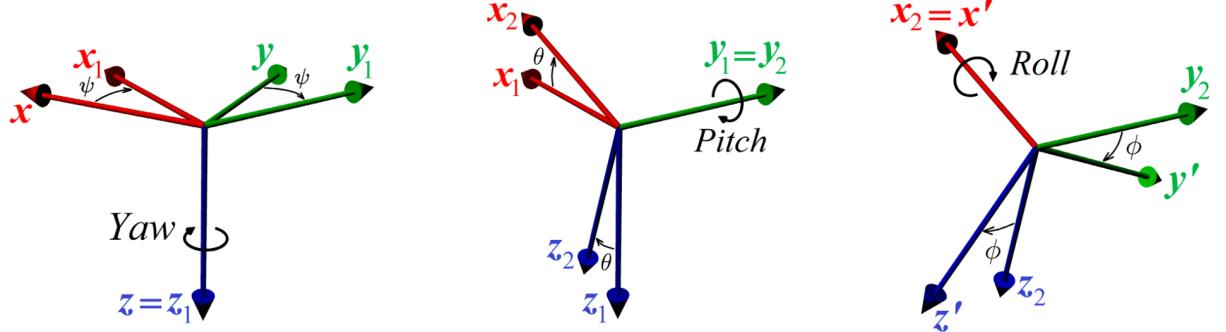


Figure 2.5: Rotation about the three axes in the order z-y-x [56]

These rotations can be derived in the same way by looking down the axis to be rotated about. The rotation about each axis is then achieved by setting the corresponding element to one and the rest to zero, leaving that axis unaffected. The rest of the matrix is a combination of trigonometric functions for the angle to be rotated. Thus, the rotation about the three axes can be done as follows:

$$R_z(\psi) = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}, R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix} \quad (2.4)$$

Relating the frames can be done by matrix multiplication, where the order goes from the current orientation multiplied by the rotation on the right. However, rotating a vector is done by multiplying the vector with the rotation matrix from the left. Though the three elementary rotations are highlighted in (2.4), they can be multiplied together and stored as one  $3 \times 3$  matrix with just 9 parameters. The simplicity of the transformations consisting of matrix multiplication is a major reason for the rotation matrices' popularity. Furthermore they have a couple useful features, namely the fact that the rotation can be reversed by multiplying its transpose [132]. Though a rotation matrix has more parameters than other formalisms, it is unambiguous as there is exclusively one rotation matrix for a specific orientation in  $R^3$  [102].

### 2.2.2.2 Euler Angles

Euler angles are a way to express rotation in  $R^3$  as a combination of elementary rotations about the three axes. As coordinate frames are defined with three basis vectors, there are  $3 \cdot 3 \cdot 3 = 27$  different combinations; though rotating about the same axis twice without another axis between would prove redundant, there are only  $3 \cdot 2 \cdot 2 = 12$  combinations. These twelve combinations fall under two categories; proper Euler angles and Tait-Bryan angles. The former conceptualizes the rotations as extrinsic (the axes of rotation don't change) and is only about two axes, with one of them being used twice:

x-y-x, y-x-y, x-z-x, z-x-z, z-y-z, y-z-y

The latter category does the consequent rotations about the intrinsic frame and uses all three axes:

x-y-z, x-z-y, y-z-x, y-x-z, z-y-x, z-x-y

This is shown in figure 2.5, where the next rotation is about the previously rotated axes.

The Euler angles have the advantage of only being stored as the three angles as parameters, though Euler rotations would require the use of multiple rotation matrices, thus taking more space and complexity. The Euler angles can also experience a problem called "gimbal lock", where two of the axes of rotation lines up, resulting in losing one degree of freedom. There are certain methods tackling this issue, though requiring more resources. As there are a multitude of compound rotations which represent a 3-dimensional orientation, this also has to be addressed to avoid ambiguities.

### 2.2.2.3 Rotation Vector (Euler Axis and Angle)

Another representation of orientation in three dimensions is the rotation vector. A unit vector  $\hat{e}$  represents the axis to be rotated about, multiplied by a scalar  $\theta$  proportional to the angle about the axis.

$$r = \theta \hat{e} = \theta \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \quad (2.5)$$

This is a simple representation with only three parameters, providing more compact attitude data. The downside with this form is that it can be complicated to perform calculations with this representation, such as multiple rotations between the reference frames. To do such, it is recommended to instead convert to another formalism such as rotation matrices or quaternions, do the calculations and convert back to the angle-axis form.

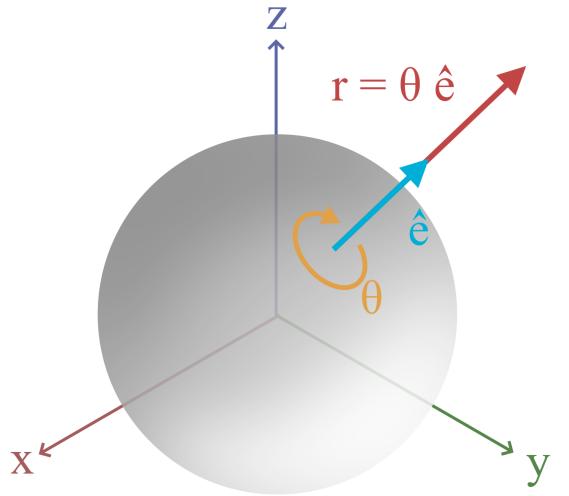


Figure 2.6: Rotation vector, also called axis-angle representation

### 2.2.2.4 Quaternions

Quaternions give a four-dimensional notation representing the orientation in  $R^3$ , using one real part and three imaginary parts  $i$ ,  $j$  and  $k$ . It can be represented in multiple ways, though a more intuitive mathematical notation utilizes trigonometric functions:

$$q = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)(ai + bj + ck) \quad (2.6)$$

Though different to work with, they also provide information about the axis of rotation and angle to be rotated, much like the rotation vector shown in figure 2.6. The axis of rotation is stored as a unit vector with the elements  $a$ ,  $b$  and  $c$ . The angle is divided by two inside the trigonometric functions, due to the nature of how quaternions work. As the rotation is a result of 4-dimensional hypersphere being stereographically projected onto our 3-dimensional space, one rotation in  $R^4$  is not equal to one rotation in  $R^3$ . Thus, a specific orientation in 3D space can both be achieved by direct rotation and double rotation as they have different orientations in  $R^4$ . This ambiguity when expressing 3-dimensional objects is the primary downside to quaternions, other than their complexity. Despite these disadvantages, quaternions store the rotation with only four parameters and are computationally effective. Furthermore, as they take place in a higher dimension, an arbitrary rotation can be expressed as a single movement (contrary to Euler angles), and therefore provides smooth movement which can be interpolated. A good intuition of how quaternions behave can be made through the explorable video series by Grant Sanderson and Ben Eater [103].

## 2.3 Fiducial Markers

For localization of objects in three-dimensional space, a multitude of solutions exist. In order to facilitate the state estimate process, fiducials (artificial features) can be applied in order to make detection and tracking simpler.

Amongst them is the ARTag (Augmented Reality Tag) marker system which was created by Mark Fiala, with improvements from its predecessor ARToolkit. *"ARTag markers are bi-tonal planar patterns containing a unique ID number encoded with robust digital techniques of checksums and forward error correction (FEC)"* [38]. Used in a variety of different applications ranging from augmented reality to robot navigation, they provide not only detection and tracking of the desired object, but can also store over two thousand unique IDs. Two newer well-known fiducial markers are AprilTag and ArUco. The former was developed by researchers at University of Michigan with support for C, C++ and Python. The latter was created by spanish researchers Sergio Garrido and Rafael Muñoz [44], along with a library for OpenCV made in C++ with support for Python [85]. The name is a combination of the original application and origin, where "ArUco" stands for "Augmented Reality University of Cordoba". Both AprilTag and ArUco can be used with OpenCV under the ArUco module.



Figure 2.7: ARToolkit Marker [62]

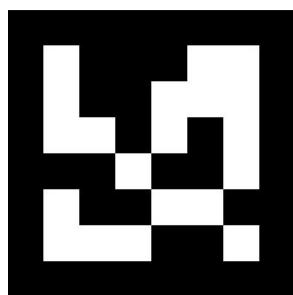


Figure 2.8: ARTag Marker [99]

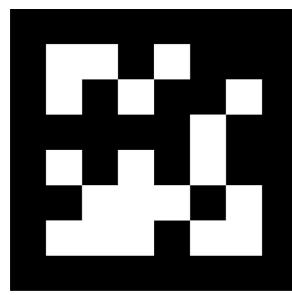


Figure 2.9: AprilTag Marker [39]

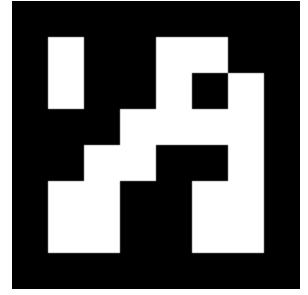


Figure 2.10: ArUco ID 306, 6x6 dictionary [59]

Though ARToolkit and ARTag have robust algorithms, both AprilTag and ArUco provide more open-source support. There are a multitude of other markers on the market such as CanTag, MaxiTag, Fourier Tag and many more, though not as easily available through OpenCV. QR-codes (Quick-Response) also include some fiducial markers, but primarily used for storing information rather than tracking. A common factor for many of these competitors are more advanced algorithms, with consequently longer computation times. Suitable for dynamic real-time applications, simpler markers may be more desirable.

The markers supported by OpenCV utilizes one of the most popular approaches, namely a binary square field. Using only black and white assures a significant contrast for a more robust identification. The marker is detected by the black outer edge, then the rest of the marker is identified inside. The white squares represent an unique ID, as well as allowing error detection and correction. Furthermore, they are rotationally unique so the orientation can also be identified through machine vision. There are multiple configurations for the markers, referred to as dictionaries, which define the internal matrix with white squares. Larger dictionaries lead to longer computation times due to the increased number of stored bits [85], but can provide more information and can lead to more robust tracking. The size decision will impact the performance and should therefore be considered before implementing into projects, as different systems have varying criteria.

## 2.4 Machine Vision

Machine vision is a systems engineering discipline for automated visual processes, where it is important to make the system perceive the images correctly and take actions based on what it is sensing. Contrary to image processing, machine vision focuses on extracting relevant information from the image and perception algorithms. Visual automation is used for many various robotics applications such as autonomous vehicles like drones, self-driving cars and robotic arms for production. Another widely known application is visual inspection in production, where machine vision is used for controlling if products are satisfying the given standard [127].

### 2.4.1 Intrinsic Parameters

The first step for machine vision applications is to have a physically correct image. With an incorrect representation of the real world, performing actions based on the analysis could be catastrophic. Therefore it is important to process the image for a most correct representation possible. There are multiple sources for making the image differ from the real world, such as external factors like lighting which can be combated with solutions like projecting structured light. However, the most important reason for an incorrect representation is due to the cameras characteristics. The primary intrinsic parameters are the intrinsic camera matrix (also referred to as the camera calibration matrix, or simply camera matrix) and distortion coefficients. The intrinsic camera matrix is a representation of the cameras focal length ( $f_x, f_y$ ) and optical center ( $c_x, c_y$ ):

$$\text{Intrinsic Camera Matrix } A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.7)$$

This is the standard form of the camera matrix, where the focal lengths represent scaling and the optical center is included for translating the origin. The camera matrix is derived in chapter 6 of "Multiple View Geometry in Computer Vision" by Hartley and Zisserman [48]. Furthermore, the camera calibratioon matrix may include a skew parameter, though it will be zero for most normal cameras according to Hartley and Zisserman [48]. Geometric distortion is also part of the intrinsic parameters, often represented as a vector of coefficients (2.8):

$$\text{Distortion Coefficients} = [k_1 \ k_2 \ p_1 \ p_2 \ k_3] \quad (2.8)$$

Where  $k_n$  and  $p_n$  are the  $n^{th}$  radial and tangential distortion coefficients, respectively. This vector can then be used for the Brown-Conrady model, with the radial distortion representation (2.9) and tangential representation (2.10).

$$x_{distorted} = x(1 + k_1r^2 + k_2r^4 + k_3r^6) \cdot y_{distorted} \quad (2.9)$$

$$x_{distorted} = x + [2p_1xy + p_2(r^2 + 2x^2)] \cdot y_{distorted} \quad (2.10)$$

Where  $r$  is the distance between the distortion center and distorted points [82]. It should be noted that this representation of the Brown-Conrady model is a simplified version withut higher-order coefficients, only representing radial and tangential distortion. Some cameras have lenses which cause substantial distortion, primarily radial and tangential but also other kinds such as prism distortion.

There are mainly three known radial distortions; barrel distortion is when the points tend to move outward whilst pincushion distortion is when they tend to move inward. A combination of the two is sometimes referred to as mustache distortion, as horizontal lines bulge outward from the middle and the other direction in the ends like a handlebar mustache. Tangential distortion refers to when

the image becomes twisted, due to the lens not being parallel to the image plane. Radial distortion however, is from the light bending incorrectly due to the shape of the actual lens.

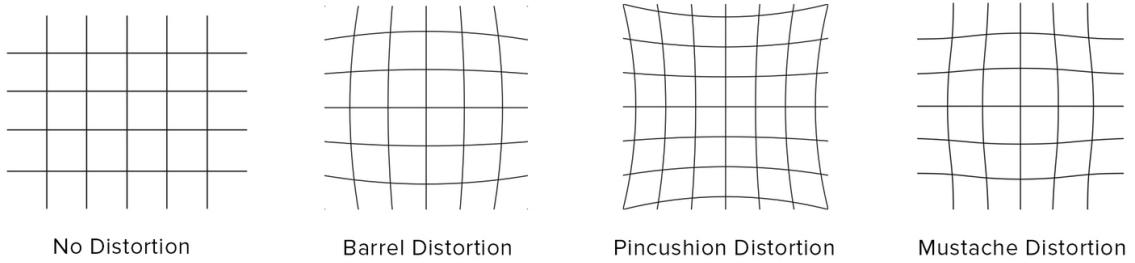


Figure 2.11: Comparison of different radial distortions [46]

With the cameras intrinsic parameters, one can reverse the distortion (called undistortion) for a more correct image. Furthermore, it allows for transformation between the cameras two-dimensional coordinates and the three-dimensional coordinates of captured objects. Further image processing may often be advisable, in order to extract the desired information from the image. Processing the lighting levels, contrast and colors (often reduced to only a single light color, called grayscale) is commonly done to reduce the total information for easier extraction of relevant characteristics. Specifically grayscale is well known to simplify the analysis algorithms. Many mathematical methods have been developed for machine vision such as convolution, contour detection and matching.

## 2.4.2 Camera Geometry

### 2.4.2.1 Homogenous Coordinates and Extrinsic/Intrinsic Camera Matrices

Homogeneous coordinates are different to Cartesian coordinates, but have many benefits; especially for computer graphics, machine vision and projective geometry. They allow points infinitely far away to be projected onto a plane with finite coordinates, with the added benefit of preserving parallel lines through affine transformations expressed as linear homogeneous transformations [83]. Central operations are rotation, translation, scaling and perspective projection; all possible with one simple matrix multiplication. To convert from Cartesian coordinates to homogeneous coordinates, an extra dimension is added:

$$\begin{array}{ll} \text{2D: } & \begin{bmatrix} x \\ y \\ w \end{bmatrix} \rightarrow \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} \\ \text{3D: } & \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} \end{array} \quad (2.11)$$

Likewise, it can be reversed from homogeneous to Cartesian coordinates as follows:

$$\begin{array}{ll} \text{2D: } & \begin{bmatrix} x \\ y \\ w \end{bmatrix} \rightarrow \begin{bmatrix} x/w \\ y/w \end{bmatrix} \\ \text{3D: } & \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix} \end{array} \quad (2.12)$$

Where  $w$  is a non-zero scalar, usually set equal to one. Self division by this scalar is part of the aforementioned perspective projection, which will be covered in the rest of this chapter. The transformations in the rest of this chapter are based on OpenCV's documentation [83] and the works of Hartley and Zisserman [48].

### Camera Coordinates (3D) to Normalized Coordinates in Image Plane (2D)

A  $3 \times 4$  projective transformation matrix can be used to map 3-dimensional points in camera coordinates to 2-dimensional ones in the image plane

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (2.13)$$

where  $x' = X_c/Z_c$  and  $y' = Y_c/Z_c$  for representing them in normalized camera coordinates [83].

### World Coordinates (3D) to Camera Coordinates (3D)

A point in the world coordinate frame can be represented in the camera coordinate frame through rotation  $R$  and translation  $t$  with a homogeneous transformation [83]:

$$P_c = [R | t] P_w = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_w \quad (2.14)$$

$\Downarrow$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Where  $[R | t]$  is called the extrinsic camera matrix, and can be both represented as a  $4 \times 4$  square matrix in homogeneous coordinates and as a  $3 \times 4$  matrix as shown below.

### World Coordinates (3D) to Normalized Coordinates in Image Plane (2D)

*"Combining the projective transformation and the homogeneous transformation, we obtain the projective transformation that maps 3D points in world coordinates into 2D points in the image plane and in normalized camera coordinates"* [83]

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (2.15)$$

### Both Intrinsic and Extrinsic Camera Matrices

Finally, the equations can be combined with the intrinsic matrix  $A$  for a more accurate mapping:

$$s p = A [ R \mid t ] P_w \quad (2.16)$$

$\Downarrow$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Where the pixel coordinates  $(u, v)$  has to be divided by the  $s$  component of the resulting vector to convert to Cartesian coordinates in the plane. This means that even if the products on the right have been scaled by an arbitrary factor, the resulting pixel coordinates would remain unaffected due to the normalization. The intrinsic and extrinsic camera matrix can be expressed as one, namely the camera projection matrix:

$$P = A [ R \mid t ] \quad (2.17)$$

where  $A$  is the intrinsic camera matrix and  $[ R \mid t ]$  is the extrinsic camera matrix. This camera projection matrix can be expressed as an unknown  $3 \times 4$  matrix:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \quad (2.18)$$

#### 2.4.2.2 Camera Calibration: Zhang's Method

In order to achieve an image which correctly represents the real world, calibration is highly advisable for estimating the cameras parameters. There are a multitude of different calibration methods, but they mostly revolve around the same principle. In this section, Zhengyou Zhang's method will be explored as it is a well known calibration technique used with OpenCV [71].

Start by obtaining a physical chess board with known dimensions, which can either be printed on paper or displayed on digital screen. There are multiple kinds of calibration patterns like the classic chessboard with black squares, asymmetrical and symmetrical circles, as well as a chessboard with fiducial markers for more robust calibration (as the classical chessboard may be more prone to occlusions) [86] [133].

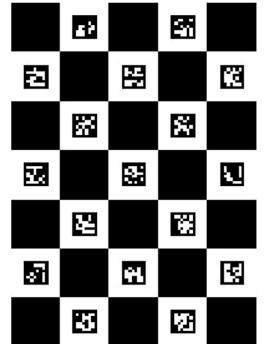
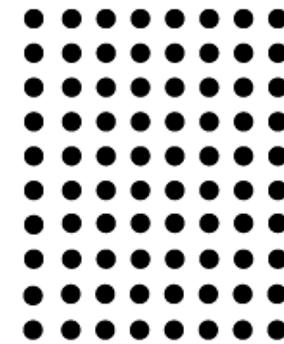
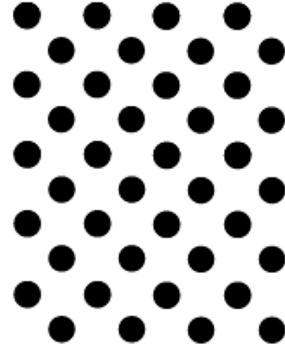
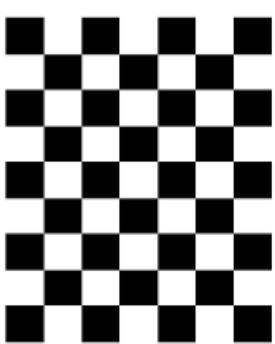


Figure 2.12: Patterns: Classic, asymmetrical and symmetrical circles [53]

Figure 2.13:  
"Chess-board +  
ArUco = ChArUco"  
[86]

When capturing images with the camera, two coordinate frames will be emphasised; the camera and the world. The  $n$  number of points captured with the camera have the following relationship with the corresponding world points, represented with homogeneous coordinates:

$$s p = A [R \mid t] P_w = P P_w \quad (2.19)$$

$$= s \begin{bmatrix} u^n \\ v^n \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X_w^n \\ Y_w^n \\ Z_w^n \\ 1 \end{bmatrix}$$

The world coordinate systems origin may be arbitrarily placed, where it is convenient for camera calibration to leave it on the planar surface being captured. This leads to the Z-component of the world coordinate system equaling zero, also cancelling out the third column in the camera projection matrix when performing matrix multiplication with homogeneous coordinates:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (2.20)$$

With the third column of the camera projection matrix removed, it is now a  $3 \times 3$  matrix which is referred to as the homography matrix  $H$ . When images are captured of the same planar surface and related through this homogeneous transformation, it is called homography which is also used for other things such as image rectification. Due to the normalization addressed in section 2.4.2.1, the last parameter  $H_{33}$  in the homography matrix is set to one:

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & 1 \end{bmatrix} \quad (2.21)$$

With  $n$  captured images of the planar surface, equations can be stacked in a homogeneous system of linear equations. The homography matrix  $H$  can then be estimated through solving the system of linear equations with singular value decomposition (SVD) as long as the planar surface is captured from at least three locations ( $n \geq 3$ ) [133]. With an increasing amount of captured images, the relative error tends to slightly decline. Though the increased accuracy past seven images is relatively low, it may be worth considering their relationship which is shown in figure 2.14.

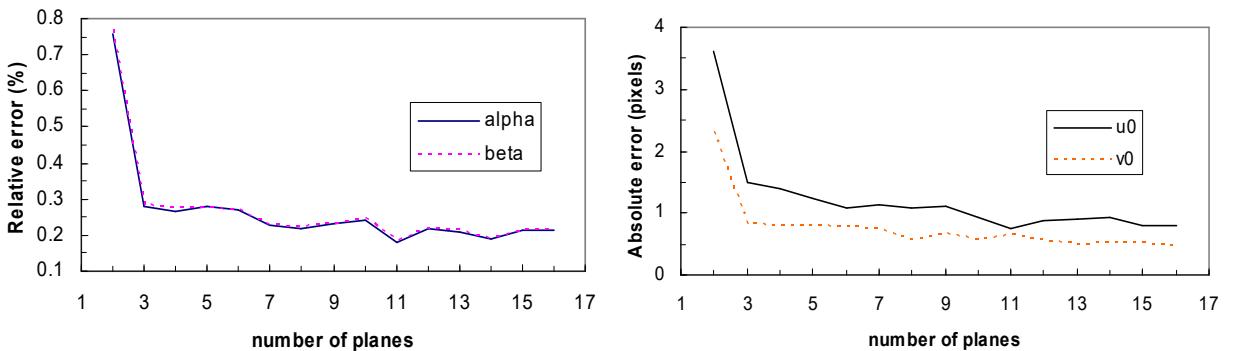


Figure 2.14: Errors versus number of images  $n$  of plane, from dr. Zhang's paper [133].

### 2.4.3 Computer Vision Library: OpenCV

The name OpenCV comes from "Open Source Computer Vision" and is a library with interfaces for C++, Python, Java and MATLAB [81]. It facilitates the machine vision process as it provides a vast amount of methods and algorithms, ranging from calibration to pose estimation with fiducial markers. Implementation in Python may be desired for development, as it is one of the simplest languages. This can be done by importing the library `cv2`, which is a newer version with improvements from its predecessor (`cv`). The programs may be transferred over to other languages such as C++ for further optimization in case the program lacks performance, a feasible task provided that the person has knowledge of the syntax for the respective languages.

#### 2.4.3.1 OpenCV: Camera Calibration

OpenCV utilizes Zhang's method for camera calibration (section 2.4.2.2) [71], with the homogeneous relations described in section 2.4.2.1. This can primarily be done with the `calibrateCamera()` function, though requiring some setup beforehand. It mainly takes in three arguments; the world points in 3D, the image points in 2D and the camera resolution. The image points can be obtained through the `findChessboardCorners()` function, which identifies the chessboards vertices and returns the coordinates of the corners. The world points are known, as the coordinate is set on the planar surface, so the calibration surface's physical attributes can be appended to an expanding array for the function's argument. The `calibrateCamera()` method has multiple outputs, with the camera matrix and distortion coefficients being mandatory [87]. The default distortion coefficients are represented as a vector in the form shown in (2.8) representing radial and tangential distortion as in equations (2.9) and (2.10).

The camera calibration process is often paired with the sub-pixel accurate corner locator for a more accurate result. It utilizes the algorithm by Förstner and Gülich [41] for detection of more accurate sub-pixel locations of radial saddle points ("saddle point" due to the formation made when zooming in on the pixel intensity array) [87]. It works by iterating through the points of a small window around the identified point (in this case a corner). It identifies the location of the sub-pixel coordinate by minimising the error  $\epsilon_i$  in the following expression:

$$\epsilon_i = DI_{p_i}^T \cdot (q - p_i) \quad (2.22)$$

Where  $DI_{p_i}$  is a pixel intensity gradient at the analyzed point  $p_i$  in the neighborhood of  $q$ , shown as red vectors in figure 2.15. This can be rewritten to the error  $\epsilon_i$  equal zero and moving the negative gradient term to the other side.

Further rewriting the gradient terms respectively to  $G$  and  $b$  can be used to express a function of  $q$ :

$$\sum_i (DI_{p_i} \cdot DI_{p_i}^T) \cdot q = \sum_i (DI_{p_i} \cdot DI_{p_i}^T \cdot p_i) \quad \Rightarrow \quad q = G^{-1} \cdot b \quad (2.23)$$

Using these equations, the algorithm iterates through the search window until the error  $\epsilon_i$  is minimized. This means the vector from point  $p_i$  to  $q$  is the most orthogonal to the neighboring gradients, meaning it will reveal the true gradient, thus more accurately representing the corner coordinate.

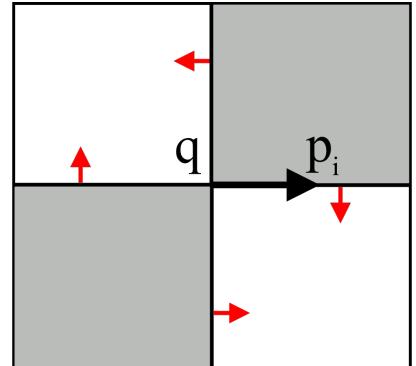


Figure 2.15: Sub-pixel Accurate Corner Locator. Figure is inspired by OpenCV's documentation. [87]

This can be done with OpenCV's `cornerSubPix()` method, which uses pass-by-reference to change them and send them out again. It also requires four more input arguments; the image to do the algorithm on, the size of the search window, a zero zone and termination criteria. The zero zone is a parameter for disabling the detection inside a part of the search window to prevent singularities, which can be disabled by being set to  $(-1, -1)$ . The termination criteria should also be set up with a desired tolerance and maximum number of iterations.

#### 2.4.3.2 OpenCV: Pose Estimation

Pose estimation of an object may be performed with fiducial markers through the use of the `estimatePoseSingleMarkers()` method in combination with the `detectMarkers()` method, both from OpenCV's aruco module.<sup>1</sup> The latter function can be used on a grayscale image with specified dictionaries and parameters, where the parameters can be created from the class `DetectorParameters` with its constructor. The `detectMarkers()` function will then return the corner coordinates, which can be used as one of the arguments for the `estimatePoseSingleMarkers()` method. The pose estimate function also requires the real size of the marker as well as the camera matrix (intrinsic camera matrix) and distortion coefficients from as arguments. OpenCV has later replaced this function with numerous "Perspective-n-Point" methods for giving a pose estimate from a set of points. These methods estimate the pose from 3D-2D correspondence, and require a setup similar to OpenCV's camera calibration with a 3D template for the object points [88]. Though they require more setup, more information may be extracted from them, such as multiple solutions for the current orientation and reprojection error.

When combining either of these methods for pose estimation with the marker detection, there is not always an unique solution. This is important to note regarding markers, that ambiguous situations arise due to the camera not having information about more than the projected coordinates of the observed object. This can be observed in figure 2.16, where a detected marker in the image can be oriented in multiple ways. This problem may be addressed by providing additional information for constraining the solution closest to the real orientation. A such possible solution has been proposed with fusion of information from RGB and depth sensors [57]. The ambiguity issue is most prominent when the marker is small, far away or orthogonal to the camera. It is therefore important to test with the full range of operation during development to check if this problem occurs in the implemented system.

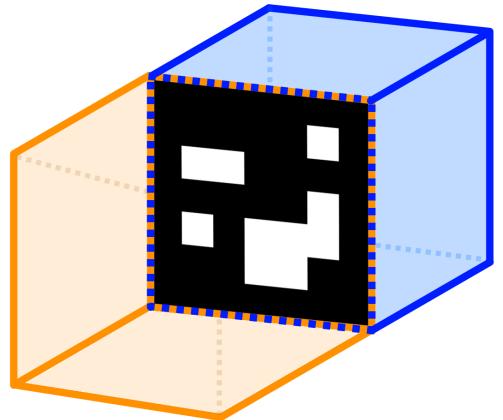


Figure 2.16: Marker orientation ambiguity, figure inspired by report on fiducial tag sensor fusion by Jin, Matikainen and Srinivasa [57]

---

<sup>1</sup> These methods are now listed as deprecated, but still work. Documentation for these can be found under version 4.6.0 of OpenCV's site [87].

## 2.5 Intel RealSense D435

Intel Realsense D435 is a solution with multiple combined components. It contains the Intel Realsense Module D430 together with a RGB Camera. The D430 Module primarily consists of an infrared projector and two wide imagers with a baseline (distance between left and right imager) of 50 millimeters. The Intel Realsense D435 also comes with its own Vision Processor D4, allowing for integrated processing [24].

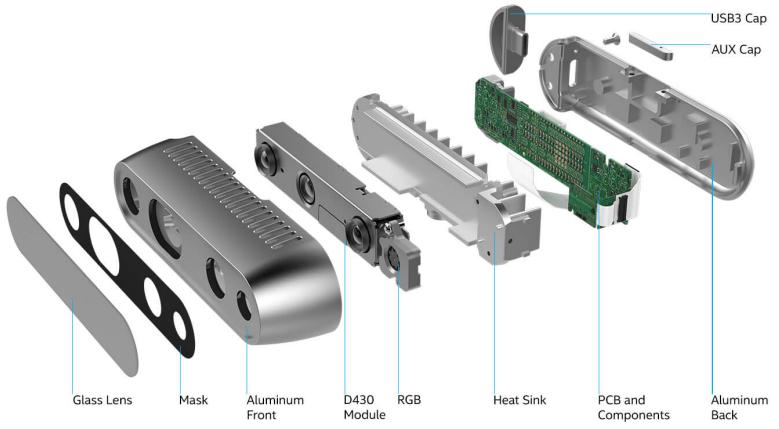


Figure 2.17: Exploded View of Intel Realsense D435 [24]

### 2.5.1 Depth Module

The D430 Module provides a left and right image sensor of type OV9282, which are monochromatic image sensors with high-speed global shutter technology. The specifications of the depth module from the D435 product documentation [24] are displayed in table 2.1.

Table 2.1: Depth Module Specifications [24] [80]

<b>Maximum Output Resolution</b>	$1280 \times 800$ [pixels]
<b>Maximum Image Transfer Rate</b>	300 fps
<b>Field of View</b>	$87^\circ \times 58^\circ$
<b>Minimum Depth Distance</b>	$\sim 28$ [cm]
<b>Focal Length</b>	1.93 [mm]
<b>Image Area</b>	$3896 \times 2453$ [ $\mu\text{m}$ ]
<b>Pixel Size</b>	$3 \times 3$ [ $\mu\text{m}$ ]

It should be noted that the maximum frames per second listed in table 2.1 is not specified in the datasheets, but throughout the project this was listed from the official SDK and confirmed while testing. The listed value for fps in the datasheet is 120 fps for the maximum resolution [80]; an expanded list with relevant values are documented in section 5.1.1. Depth is measured using these imagers through stereoscopic vision (also called stereo vision). This process measures the difference between them and records it in the whole image for generating a disparity map. The discrepancy between the imagers is inversely proportional to the real depth and can be used to map the real distance from the D430 module to the captured object.

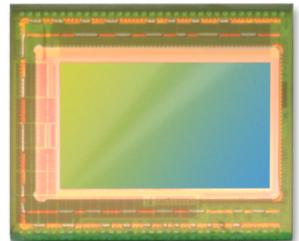


Figure 2.18: OV9282 B&W Image Sensor [80]

### 2.5.2 RGB Camera

The D435 model also comes with its own full high-definition (FHD) color sensor of type OV2740 with rolling shutter technology. The specifications from the product documentation for the color sensor (also referred to as the RGB camera or color module) are shown in table 2.2 [79] [24]. It should be noted that also the maximum fps rate listed in table 2.2 is also found through experiments, not the datasheet. An expanded list for this sensor's relevant rates is also documented in section 5.1.1.



Figure 2.19: OV2740 Color Sensor [79]

Table 2.2: OV2740 Specifications [24] [79]

<b>Maximum Output Resolution</b>	$1920 \times 1080$ [pixels]
<b>Maximum Image Transfer Rate</b>	60 fps
<b>Field of View</b>	$69^\circ \times 42^\circ$
<b>Focal Length</b>	1.88 [mm]
<b>Image Area</b>	$2728.8 \times 1549.8$ [ $\mu\text{m}$ ]
<b>Pixel Size</b>	$1.4 \times 1.4$ [ $\mu\text{m}$ ]

### 2.5.3 Infrared Projector

For providing additional texture, the Intel RealSense D435 also comes with an optional infrared projector to project a static IR pattern on the scene. The D430 module has a dedicated driver for controlling the IR laser inside the projector module. The IR projector is compliant with the Class 1 IEC 60825-1:2014 Edition 3 laser safety standard, given normal operation inside the Intel Realsense D435 camera [24]. The IR projector uses a vertical-cavity surface emitting laser (VCSEL), with many benefits over conventional edge-emitting lasers. Its low power-consumption, combined with the ability for high-speed modulation with low current and large 2D arrays are significant advantages for real-time applications such as robotics and drones [63].

The given wavelength in table 2.3 is a nominal value at a temperature of  $20^\circ\text{C}$ . For safety reasons the D430 module is also equipped with a temperature transmitter to automatically monitor the integrated IR projector, which can be read through the software development kit. If the temperature exceeds  $60^\circ$  while streaming, power is halved to the laser [24]. Furthermore, the laser will turn off if the temperature does not go below a threshold within a given timeframe.

Table 2.3: Infrared Projector Specifications [24]

<b>Maximum Output Resolution</b>	$1280 \times 800$ [pixels]
<b>Maximum Image Transfer Rate</b>	300 fps
<b>Field of Projection</b>	$90^\circ \times 63^\circ$
<b>Mean Optical Power</b>	360 [mW]
<b>Laser Wavelength</b>	$850 \pm 10$ [nm]

#### 2.5.4 Vision Processor D4

The RealSense D435 camera comes with its own integrated processor. It can communicate with the host processor through either USB 3.1 Gen 1 or the mobile industry processor interface (MIPI: "a global, open membership organization that develops interface specifications for the mobile ecosystem"[24]), though universal serial bus (USB) is most common and may be preferred. It also supports the serial peripheral interface (SPI) or inter-integrated circuit (I2C) protocol, as well as general purpose IO pins [24]. The Vision Processor D4 works as an interface between the different camera modules and the host processor, as shown in figure 2.20. The vision processor can also perform image rectification and control the IR laser.

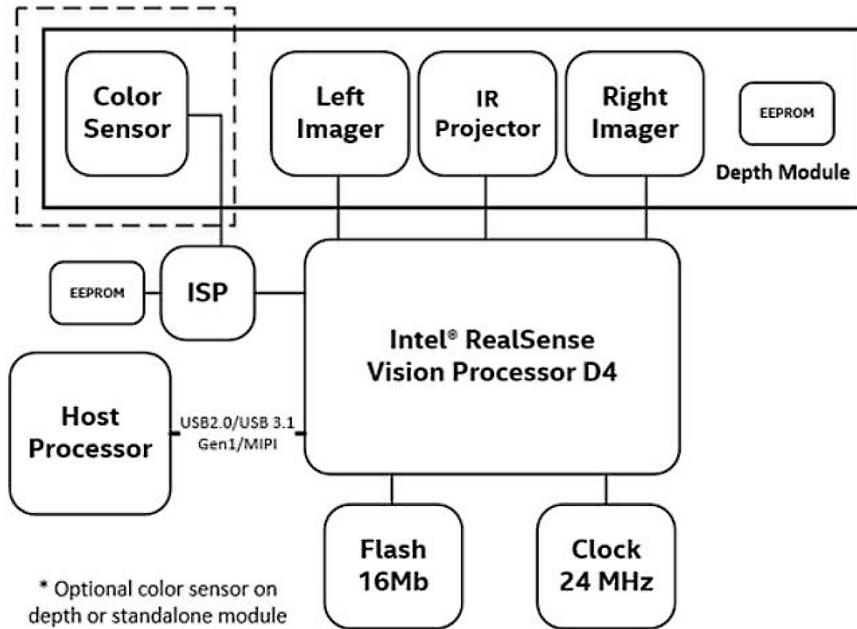


Figure 2.20: Vision Processor D4 Camera System Block Diagram [24]

## 2.6 Wrappers

Wrapper libraries are a special kind of library, which work as an interface between the user and other existing libraries. They turn the existing interface more abstract, making them simpler and more efficient to work with. `pyrealsense2` is such a library, since it works as an interface between the Python code and the more low-level C++ code from the SDK (for instance used for controlling the Intel RealSense D435 camera). The wrapper provides pipelines for transmitting data, which can take custom configurations such as which module to read (depth or color), resolution, fps and more. The wrapper also includes a method for time-synchronised frames of every stream that is configured [116].

## 2.7 Statistics

Prior to approaching the statistical applications in this report, it is important to have a firm grasp of the central concepts. With an arbitrary dataset  $x$ , one can calculate many weighted measures to quantify the characteristics. To keep this section brief, only the relevant terms will be explained. The mean of the dataset is the sum of the values  $\Sigma x$  divided by the number of elements  $n$

$$\mu = \frac{\Sigma x}{n} \quad (2.24)$$

The average  $\mu$  gives an indication of the expected value of a new number appended to the dataset.

Other than this, a measure of the spread is desired. This can be done with the population variance  $\sigma^2$  calculated with the total variation  $SS_x$

$$\sigma^2 = \frac{SS_x}{n} = \sum_{k=1}^n (x_k - \mu)^2 \div n \quad (2.25)$$

From this, the standard deviation  $\sigma$  is simply the square root of the variance  $\sigma^2$ .

A probability distribution function (PDF) is a function which can model the likelihood of the values in a dataset. There are a multitude of different PDFs, which take different forms and can be either continuous or discrete. Common for them is that they have probability values along the function between 0 and 1, with the sum of the probability axis equal to one, that is 100% of the data.

Curve fitting is a mathematical model approximated from a dataset. To quantize how well this model fits the data, two well known statistics can be calculated: R-square and root mean squared error (RMSE). The R-squared is a ratio of the "Sum of Squares of Regression" (SSR) and the "Total Sum of Squares" (SST), which gives information of how well the model fits the data [121]. It is calculated as follows:

$$R\text{-square} = R^2 = \frac{SSR}{SST} = \frac{\sum_{i=1}^n (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (2.26)$$

The ratio goes from 0 to 1, where a value close to one means the model accounts for a greater proportion of variance [121]. The RMSE can be calculated as the square root of the residual mean square, which is the "Sum of Square Error" (SSE) divided by the number of non-missing data points  $n$ , as shown in (2.27). RMSE values closer to 0 indicate a better fit with less residuals.

$$RMSE = \sqrt{\frac{SSE}{n}} = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}} \quad (2.27)$$

### 2.7.1 Normal Distribution

Central to statistics is the Gaussian distribution. The characteristic bell curve is so common and well known, that the distribution has been named "The Normal Distribution" [78]. It appears almost everywhere, from abstract concepts like intelligence quota (IQ) to sensor noise. The normal distribution can be modeled with the two parameters  $\mu \in \mathbb{R}$  and  $\sigma > 0$  (meaning that the expected value is a real number and the standard deviation is more than zero), with the following function:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.28)$$

The notation for expressing the variable  $X$  as a parameter described by the normal distribution is done with the symbol  $\mathcal{N}$  (also noted with the symbol  $\phi$  outside this thesis):

$$X \sim \mathcal{N}(\mu, \sigma) \quad (2.29)$$

One can observe how the normal distribution is modeled with combinations of various expected values  $\mu$  and standard deviations  $\sigma$  in figures 2.21 and 2.22.

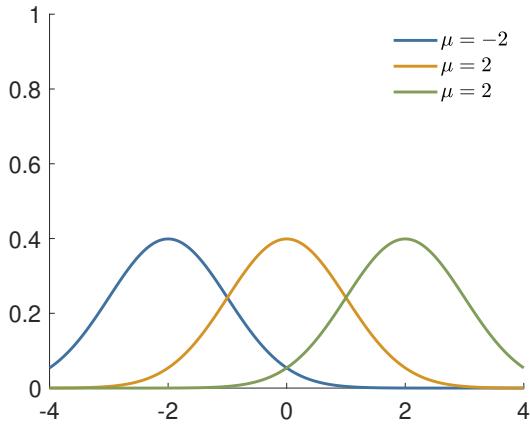


Figure 2.21: Distributions with  $\sigma = 1$   
MATLAB simulation inspired by Nyberg [78]

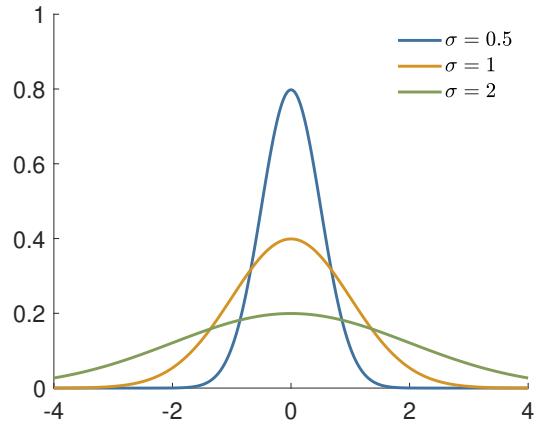


Figure 2.22: Distributions with  $\mu = 0$   
MATLAB simulation inspired by Nyberg [78]

### Zero Mean White Noise

Relevant to this thesis is sensor noise, which is commonly modeled as "zero mean white noise"

$$n_k \sim \mathcal{N}(0, \sigma) \quad (2.30)$$

It is a normal distribution with noise  $\mu = 0$ , as shown in (2.30) and reflected in the "zero mean" part of the name. The term "white noise" refers to the fact that the values are spread randomly at different frequencies (much like white light, which has wavelengths at different frequencies).

## 2.8 State Space Representation

A physical time-varying system may be represented with a mathematical model called state space, which is represented by equations (2.31) and (2.32)

$$\dot{x} = Ax + Bu \quad (2.31)$$

$$y = Cx + Du \quad (2.32)$$

with the mathematical notations explained in table 2.4. The model gives information of the next state of the system, based on the previous state and current inputs. The system state vector varies from system to system, but can be used to express various systems such as the positions and velocities of a mass-spring-damper system or the behaviour of an electrical system [76].

Table 2.4: State Space Variables

Symbol	Description	Dimensions
$x$	State Vector	$n \times 1$
$u$	Input/Control Vector	$m \times 1$
$y$	Output Vector	$p \times 1$
$A$	System/State Matrix	$n \times n$
$B$	Input Matrix	$n \times m$
$C$	Output Matrix	$p \times n$
$D$	Feedthrough/Feedforward Matrix	$p \times m$

## 2.9 Kalman Filter

The Kalman filter, also referred to as linear quadratic regulator (LQR), is an optimal estimator designed for stochastic systems. The filter is designed to work well with state space models where it estimates the state of the system, by observing the measured outputs  $y$  and the control input  $u$  over time. Real systems have external factors that are unpredictable to a certain degree, therefore neither the mathematical model nor the sensors are adequate on their own. Sensors are always subjected to inaccuracies in some way, such as statistical noise and bias.<sup>2</sup> The filter models the measurement noise and process noise of the prediction model as Gaussian distributions with zero mean.

Various different state estimators exist, such as the continuous and discrete versions of the Kalman filter, the extended Kalman filter (EKF), unscented Kalman Filter (UKF) and particle filter (PF). This report will focus on the discrete implementation of the Kalman filter. As the filter incorporates the state space model, this chapter will expand upon the mathematical notations shown in table 2.4. Consider a discrete-time linear system with the following state space model:

$$x_k = A_k x_{k-1} + B_k u_k + v_k \quad (2.33a)$$

$$y_k = C_k x_k + w_k \quad (2.33b)$$

Where the measurement noise  $v_k$  and process noise  $w_k$  are zero mean Gaussian distributions with covariance matrices  $R_k$  and  $Q_k$ :

$$v_k \sim \mathcal{N}(0, R_k) \quad w_k \sim \mathcal{N}(0, Q_k)$$

The corresponding Kalman filter for this system consists primarily of two steps; prediction and update.<sup>3</sup> The probability density functions are shown in figure 2.23, illustrating that the estimated state  $\hat{x}_k$  is a combination of the predicted state estimate  $\hat{x}_k^-$  and measurement  $y_k$ .

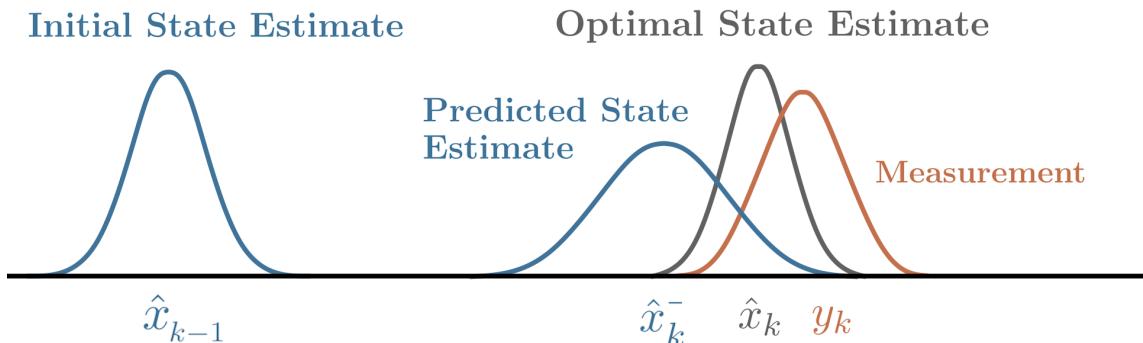


Figure 2.23: Probability density functions for Kalman filter algorithm [122]

**The prediction step** calculates the a priori estimate  $\hat{x}_k^-$  and the predicted error covariance matrix  $P_k^-$ :

$$\hat{x}_k^- = A_k \hat{x}_{k-1} + B_k u_k \quad (2.34a)$$

$$P_k^- = A_k P_{k-1} A_k^T + Q_k \quad (2.34b)$$

Where the predicted state estimate  $\hat{x}_k^-$  has variance  $P_k^-$  which comes from the process noise and propagation of the uncertain previous state. **The update step** uses the a priori estimate  $\hat{x}_k^-$  and error covariance  $P_k^-$ , and estimates the posterior state estimate  $\hat{x}_k$  and error covariance  $P_k$ :

<sup>2</sup>Bias is an expression referring to a systematic error resulting in a fixed offset in the measured value.

<sup>3</sup>Also called propagation and measurement steps.

$$K_k = P_k^- C^T (C P_k^- C^T + R_k)^{-1} \quad (2.35a)$$

$$\hat{x}_k = \hat{x}_k^- + K_k (y_k - C \hat{x}_k^-) \quad (2.35b)$$

$$P_k = (I - K_k C) P_k^- \quad (2.35c)$$

The Kalman gain  $K_k$  is updated each iteration, where the measurement estimation error  $\tilde{y}_k$  is scaled corresponding to the uncertainty of the current prediction, to minimize the posterior error covariance  $P_k$ .

$$\tilde{y}_k = (y_k - C \hat{x}_k^-) = (y_k - \hat{y}) \quad (2.36)$$

As shown in equation (2.35a), a high uncertainty in the measurements means the measurement noise covariance  $R_k$  is high, which makes the Kalman gain weight the latter term less in equation (2.35b), thus trusting the prediction model more. The opposite is true, with  $R_k$  being small. It should be noted that depending on the system, the covariance matrices  $R_k$  and  $Q_k$  can either consist only variances or also covariances. In the former case, the variances should be placed along the diagonal of the matrix, to correspond to the vector variables. This situation may arise when the variables can be assumed to not have any direct correlation, and therefore be modeled with individual noise. Caution is advised to ensure the use of variance and not standard deviation, as the latter is the square root, resulting with an incorrect noise model. As shown, the discrete Kalman filter algorithm is a relatively compact algorithm which can be computationally effective. The filter is a well known control algorithm, due to the capability to model systems even in unpredictable environments with its probabilistic nature. The filter can also be modified with logic so it updates the estimate even when the system is not measured, enabling the possibility to give a state estimate with higher frequency than the sample rate from the sensors of the physical system. This property makes it desirable for real-time applications, where response time is critical. There are some downsides to the standard Kalman filter algorithm, as it is only capable of modeling linear systems with zero mean Gaussian distributions. The EKF can linearize systems which are locally linear, though if the system is completely non-linear, other sigma-point filters may be desired. They are less computationally effective, but can extend the functionalities, such as the particle filter's ability to model other distributions than Gaussian. Some filters can also estimate the errors as separate states, which can be used to remove bias.

## 2.10 Software Development

### 2.10.1 Object oriented programming and polymorphism

Object Oriented Programming (OOP) is defined by the usage of user defined types and the inheritance of these type to provide a more modular code. These types have their own operations defined for its purpose, and an instantiation of a type is called an object [110]. The types also allow for data-hiding, since if the data is not to be used outside of the operations of the type then it can be hidden from the rest of the program. The hiding of data allows for data-abstraction to be used which is the concept of simplifying the usage of a complex method to only what is required [106]. In the case of C++ and other languages, OOP is tackled by using classes and structs as the type, where inheriting a class allows the derived class to re-implement specific operations in an otherwise similar type, and therefore reuse the shared common concepts. In C++, functions that are able to be re-implemented are known as *virtual* functions.

Polymorphism is the case of using derived types to perform their self-specific operations based on the base type, such as that if a base class includes a function that is required to be re-implemented for its derived classes (in C++, this is called a *pure virtual*), then calling that function will result in different behaviours depending on the derived object [104].

C++ also allows for interfaces and abstract classes, where abstract classes are classes which are unable to be instantiated due to an undefined function which is the *pure virtual* function as mentioned earlier, an interface is then a class including only undefined functions.

### 2.10.2 Threading and parallelism

A thread is a subdivision of a process that shares the same memory, threads have their own separate memory stack which is also shared within the process, and an instruction pointer which points to the location of the thread in the memory. In a single core of the CPU of a computer, processes can be concurrently ran which means that each process has a set amount of time for computation before a different process is allowed to do their own computation, this method is created by the preemptive scheduler of the operating system (OS). Although the execution of threads and separate single-thread processes are similar, they are different in that separate processes have their memory isolated, causing them to be uninterruptible with each other without the use of Inter-process communication which will be explained later. With the use of multiple cores, parallelism can be achieved, as well a combination of parallelism and concurrency [130]. Parallelism is when computation is split along multiple cores, allowing computation to happen at the same time rather than waiting for the previous instructions to finish.

Due to the sharing of memory between threads in a process, the process becomes incredibly error prone without proper handling, such as errors occurring from race conditions which happens when a value is accessed and written to at the same time, or errors from the deallocation of memory without joining the thread first. To create what is known as a *thread-safe* environment, methods such as using mutable locks, which prevent a thread from executing until the lock is released, atomic variables, which have a defined behaviour when the value is read and written to at the same time, queues, and other signalling methods can be used.

The server code used in the rapport does not use parallelism for an improvement of computational times, but as a method to provide parallel communication, therefore it has an extensive use of queuing methods for thread-safe data sharing. Queues are generally First-in-First-out (FIFO) queues, where a value can be placed at the back of the queue, while the first value of the queue can be accessed [31].

As a side note for threading in C++, in C++ 11 and up, a lambda expression can be used for creating a thread object usable within a class such as with the code:

```
threadObj = std::thread([this] ()  
{  
    threadFunction();  
});
```

A lambda expression is a method of defining a function. These lambda expression have many syntaxes however the lambda expression used in the rapport and the line above uses the syntax:

```
operator = [captures] (params) {body;};
```

In this notation, the *[captures]* clause defines usable variables from one or more objects which the lambda expression is within the scope of, these are then accessible within the method defined in the body, params refers to the parameters of the function's input, such that *operator(params)* can be called. The clause *(params)* of the lambda expression may be omitted if the function has no parameters [30].

### 2.10.3 PC timers

PCs have two methods of keeping time, these are called wall clocks and monotonic clocks. Wall clocks are based on a synchronisation with a server which has access to an atomic clock. The synchronisation method is most commonly done using the Network Time Protocol (NTP), during this synchronisation of the wall clock, the computer's local time is skewed or stepped to synchronise with the server's atomic clock. Therefore for getting accurate time measurements which are not able to be changed, monotonic timers are used, which is based on the oscillations of a crystal. The oscillations of the crystal can still change due to changes in temperature so the clock is still skewed for better synchronisation, however no jumps or steps in the monotonic time will occur.

The monotonic timer is based on an arbitrary point in time such as when the crystal is first powered, therefore that a single measurement is of no use, however by comparing two measurements of the monotonic timer, an accurate measurement between the times can be made. Wall timers are based on the time passed from the Unix Epoch of 1st January 1970, this means that a wall timer can be changed from system settings, which could give inaccurate results if comparing two wall timer measurements [89].

#### 2.10.4 Registers and bitwise operations

A register is a type of memory typically with a very specific purpose, for Integrated Circuits (IC)s, the user readable or writable (R or W) registers are often used to set options of functions or read data such as from sensors [52]. Bitwise operations are used often in combination with I2C or SPI for reading and setting values of registers, most commonly the bit shifts and the OR operator. The bit shift operator often written as `<<` or `>>` shifts the bits in a value to the left or the right, such as in the pseudo-code examples:

```
val = 0001 << 1
val: 0010

val = 1000 >> 2
val: 0010
```

values from sensors are often split into multiple 8-bit registers which are often referred to as the MSB/LSB or HIGH/LOW registers, these have to be combined in the software into a single value which can then be used to find the value of the sensor with the specified units. This is done by bit shifting the MSB byte to the left 8 times, and ORing it with the LSB byte, a typical operation may look like this:

```
uint16_t full_value = uint16_t(msb_byte << 8) | uint16_t(lsb_byte);
float sensor_value = float(full_value)/65535.0f * sensor_unit;
```

## 2.11 Communication within the drone arena system

For the drone arena system to work flawlessly, communication with and within the drone, actuator system, and the main computer which contains the RL and computer vision methods needs to be implemented.

### 2.11.1 UDP and socket programming

Berkeley sockets are an Application Programming Interface (API) used for network communication, which give an interface for use with socket types such as UNIX domain sockets and internet sockets. Sockets are able to accommodate multiple protocol types for communication such as UDP,TCP, etc [28]. UDP (user datagram protocol) is a protocol with bidirectional communication where datagrams are used, datagrams are independent messages containing data which is not broken up as part of the protocol, datagrams will not always arrive in a specific order and there is no guarantee that there is an arrival of a datagram packet [29].

Another commonly used protocol is TCP (Transmission Control Protocol) where data is guaranteed to arrive and in the correct order as bi-directional communication is established before sending the data, this bi-directional communication allows for error handling in case a packet is lost. The data in TCP packets is broken up so that if a packet is lost then only the lost section can be re-transmitted rather than having to send all the data again from the beginning [42].

An advantage to using UDP is that it requires significantly less overhead, and since UDP does not have any in-built error detection or signaling for incoming packets, the latency is much higher than with TCP. This lends itself to being good for fast data-streams, where a single lost packet can be

ignored as another packet is on its way shortly.

Network ports are virtual ports which are used to separate incoming network traffic into a specific service. Network port numbers are defined by an unsigned 16 bit number, where the first 1024 ports are known as well-known port numbers which are used for commonly used services such as HTTP, while other port numbers can be used for other program functions. Dynamic ports range from port numbers 49152 – 65535 which are ports which the computer automatically gives an unused dynamic port to a socket requesting it. Getting a dynamic port is usually done by setting the port number as 0 when binding the socket [45].

Resolving a host is the act of getting an IP address from a computer's hostname. This can either be done using a hosts file stored on the local computer which maps a hostname to a specific IP address, or through the DNS server. The DNS server keeps a track of the local server's connected hosts which it will check, if the name is not found in the local database, then a request for the hostname will be sent to the ISP's server which can check the root nameservers for it [113].

### 2.11.2 Inter-process communication

The communication between processes such as multiple executable files is called inter-process communication (IPC), IPC is often used as processes might have very separate functions but still requiring the same data, this is can also be used to communicate between two running processes written in different languages. For the specific case of the drone arena, IPC is used to communicate between the reinforcement learning process and the position calculation process for example.

### 2.11.3 Data packaging

To transfer data using UDP or other IP protocols, the data needs to be serialized and sent as a string of bytes. Before the serialization is done, the data has to be packed together which is often done with data packaging files or methods such as JSON, XML, Protobufs, etc [131]. When a serialized string of data is sent, the data can then be deserialized or parsed to reform the original serialized file including the data in the correct order or fields. This means that the data packaging methods are often language neutral which is important for future modularity. Many of the data packaging formats allow for the data to be human readable, however writing the data to a file before serializing is inefficient, therefore Protocol Buffers tackles this by serializing it directly from variables in the code [118].

### 2.11.4 SPI on Raspberry PI

Serial Peripheral Interface (SPI) is a method used for serial communication, with SPI, only one peripheral can communicate with the controller at a time. In the standard Active Low mode, the peripheral which can communicate is chosen by setting its chip select line to low while all the other chip select pins for other peripherals are set to high. Messages are then sent from the controller to the peripheral through the Serial Data Out (SDO) line, and replies from the peripheral are sent through the Serial Data In (SDI) line, this can happen simultaneously. Another line is the Serial Clock (SCK) line which is used for synchronising the sampling and shifting of the data in the SDO or SDI lines. SPI has 4 modes which change the timing of sampling and shifting data depending on the clock signal such as on the rising or falling edge of the clock signal [33]. SPI devices generally have a max transfer speed of around 10[MHz], this is significantly better compared to I2C which has a max transfer speed of 400[KHz] [15].

The raspberry pi has multiple SPI ports, SPI0 is referred to as the main SPI port, and the SPI1 and up are referred to as auxiliary SPI ports. The auxiliary SPI ports consume more processing power than the main SPI port on higher clock frequencies, however they have useful extra features which the main SPI port does not have, such as being able to change the SPI byte word size, and having an easier implementation to choose between sending the MSB or LSB of the byte first. The auxiliary SPI ports on raspberry PIs do not support mode 1 and 3 [97].

### 2.11.5 Buildroot operating system

Buildroot is a tool which generates a custom Linux operating system (OS) tailor made to the systems needs. For embedded systems a Buildroot OS facilitates reduced CPU usage and power consumption, in turn reducing hardware demands and cost. OS functionality is altered by adding and removing modules depending on what tasks the hardware is required to perform.

## 2.12 Lead screw transmission

A lead screw, also known as a power screw can be seen as an incline plane that is wrapped around a cylinder, as the cylinder rotates a lead nut is pushed upwards or downwards, by unwrapping it simple equations can be formed to describe the behaviour of the movements. The most common types of lead screws have square threads or acme threads which have a universal standard and are tapered towards the end [77].

The threads of lead screws are often defined by the distance from the start of one thread to the start of the next thread, which is called the pitch and is noted by  $p$ , the length of the threads are commonly half of the distance  $p$ , and an important value for calculations is  $d_P$  which is the diameter of the rod including the thread minus the length of the thread:  $d_P = D - \frac{p}{2}$ . Another useful value is the lead of the screw which is defined as how far the lead nut will travel in one rotation of the screw.

The screw torque required to lift a load with square threads can be found by using formula 2.37, and the equation for the required torque to lower a load is shown in equation 2.38.

$$T_{S_U} = F \frac{d_P}{2} = \frac{P \cdot d_P}{2} \cdot \frac{\mu \cdot \pi \cdot d_P + L}{\pi \cdot d_P - \mu \cdot L} \quad (2.37)$$

$$T_{S_U} = F \frac{d_P}{2} = \frac{P \cdot d_P}{2} \cdot \frac{\mu \cdot \pi \cdot d_P - L}{\pi \cdot d_P + \mu \cdot L} \quad (2.38)$$

Where:

$T_{S_U}$  = Screw with square threads torque [Nm]

$d_P$  = Diameter to center of threads (pitch diameter) [m]

$L$  = Lead [m]

$P$  = Force from load [N]

$\mu$  = coefficient of friction between the screw and nut

Depending on the system, a thrust collar may be used which introduces extra friction which is defined as  $T_C$ , however the system used in the rapport does not this method.

with ACME threads, an extra factor is added into the equation due to the angle of the tapering defined as  $\alpha$ , which is  $14.5^\circ$  in standard ACME threads. The new torque equations for lifint or lowering hte load are equations 2.39 and 2.40 respectively [77].

$$T_U = \frac{P \cdot d_P}{2} \cdot \frac{\mu \cdot \pi \cdot d_P + L \cdot \cos(\alpha)}{\pi \cdot d_P \cdot \cos(\alpha) - \mu \cdot L} \quad (2.39)$$

$$T_U = \frac{P \cdot d_P}{2} \cdot \frac{\mu \cdot \pi \cdot d_P - L \cdot \cos(\alpha)}{\pi \cdot d_P \cdot \cos(\alpha) + \mu \cdot L} \quad (2.40)$$

## 2.13 MOSFET

A MOSFET is a metal oxide semiconductor field-effect transistor, which can be seen as an electrical switch, it uses a smaller voltage input to the Gate pin to form a connection for a larger voltage line to pass from Drain to Source. It is recommended to implement a resistor to the gate input of the MOSFET, the resistor both limits the current and improves response, however when the resistor is too large, the switching time increases and with a resistor too small the system may experience damped oscillations also called ringing which can limit the effectiveness of systems requiring fast switching such as with PWM, picking an optimal gate resistance is therefore recommended which is based on the desired switching time and current [125]. The optimum gate resistance can be calculated with the formula [124]:

$$R_G = \frac{V_G \cdot t_G}{Q_G} \quad (2.41)$$

where:

$R_G$  = Gate resistor [ $\Omega$ ]

$V_G$  = Gate voltage [V]

$t_G$  = Switching time [s]

$Q_G$  = Gate charge [C]

The total gate charge for switching can be found through the manufacturers datasheet for the specific mosfet, therefore the resistance is primarily based on the desired switching time which is often only as large as a few microseconds. If there is not a discrete requirement for the switching speed, the gate resistor can be calculated with Ohm's law based on the desired current pull to the transistor [66].

## 2.14 Circuit protection and Decoupling capacitors

Over voltage protection (OVP) is when circuitry is used to protect a PCB from excessive voltage such as from electrostatic discharge. Protecting the PCB from excessive current is also a critical task as this could happen such as if a component is shorted on accident.

A reverse polarity protection diode is a standard diode placed in reverse bias between GND and VCC. This diode prevents voltage from damaging components if the current was accidentally placed in the reverse order, where VCC was placed in GND instead of VCC in VCC [60]. Since the polarity protection diode has a low resistance, the current through the diode will be high, therefore if paired with a fuse this can shut off the circuit effectively without damaging any components. A fuse is a component which breaks the connection if the current passing through the fuse is above the rated trigger current, or if the fuse experiences a temperature high enough to simulate the heat caused by a high current. A resettable fuse is a fuse which is built with a conductive polymer that experiences a phase change on high temperatures, due to the phase change, conductive paths break leading it to a sharp increase in resistance which limits the current, preventing components from getting damaged [35].

A decoupling capacitor can be used to prevent voltage spikes or voltage drops by storing energy and releasing it to the circuit in the case of a voltage drop, or dissipating it during a voltage spike, this limits noise in the system which can lead to unpredictable behaviour for VCC inputs for integrated circuits (IC)s or can produce erratic behaviour in high volt components such as motors [119].

## 2.15 Trace width

The minimum recommended trace width for PCBs based on the current can be calculated using the formula 2.42 from the IPC2221 standard for PCB design [55]

$$I = K \cdot \Delta T^{0.44} \cdot (W \cdot H)^{0.725} \quad (2.42)$$

$$W = \frac{\sqrt[0.725]{\frac{I}{K \cdot \Delta T^{0.44}}}}{H}$$

Where:

$I$  = Current through the trace [A]

$W$  = Trace width [mm]

$H$  = Copper height [mm]

$\Delta T$  = Temperature increase [ $^{\circ}\text{C}$ ]

$K$  = Constant, 0.024 for internal traces, 0.048 for external traces

Using this formula, the trace width for a specific current is mostly based on an acceptable temperature increase of the trace, as changing the other parameters is often difficult or expensive.

## 2.16 Inertial Measurement Unit (IMU)

An Inertial Measurement Unit (IMU) is a sensor which is used for measuring the movement of an object which the sensor is affixed to. IMUs generally include a gyroscope, accelerometer, and sometimes a magnetometer to achieve this. Accelerometers measure linear acceleration in the units [ $m/s^2$ ] or expressed in terms of earth's gravitational acceleration [g]. Gyroscopes measure the angular velocity, often expressed as [degrees/second], [deg/s] or [dps] [22].

There are multiple types of accelerometers however the IMU used in the project has a Micro-ElectroMechanical Systems (MEMS) based accelerometer [15]. MEMS in accelerometers measure the displacement of a mass through the change of distance between capacitive plates, these plates are often arranged in a differential pair, where as the mass moves, one side gets closer to a plate as the other gets further away, this creates a difference in capacitance between the differential pair which can then be measured. Multiple of these pairs can be put in a parallel formation to give a larger difference in the capacitance allowing the readings to be more accurate [32]. A multiple axis accelerometer can be created by having the differential pair on multiple sides, this way rather than the distances between the plates changing as the mass moves perpendicular to the plates, the area between the plates will increase or decrease as the mass shifts along the direction of the plates. This way the difference of capacitance of the sides can be measured.

A MEMS gyrometer works by detecting the Coriolis acceleration of a vibrating mechanical element, the Coriolis acceleration is an acceleration that is applied on a moving mass that is fixed along an axis on a rotating reference frame [90]. The Coriolis acceleration is proportional to the angular velocity and is in the direction of the rotational motion, this acceleration causes a change in the distance perpendicular to the capacitive plates which can be measured.

## 2.17 Fundamental Aerodynamics Relevant to The Quadcopter

Despite the fact that a propeller rotates about its own axis, the working principle remains the same as a static airplane wing. Generally, the cross sectional profile of any shape designed for aerodynamic performance is referred to as an aerofoil, or airfoil, which is defined as a streamlined body which generates more lift than drag. This draws the attention to the fundamental mechanism of flight; when an object moves through fluid, it experiences a force which can be decomposed into the two vectors named lift and drag. This is shown in figure 2.24, where the drag force is parallel to the fluid flow and the lift force is normal to it. The lift force is generated from the pressure difference over and under the aerofoil, where the fluid pushes the object up.

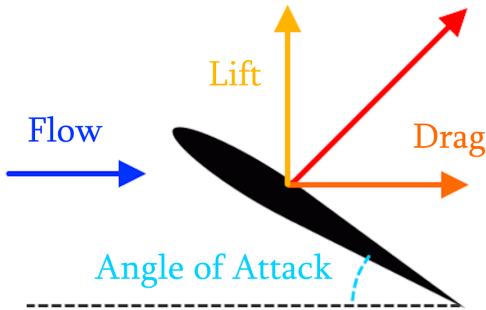


Figure 2.24: Forces generated by aerofoil moving through fluid, figure inspired by NASA [4]

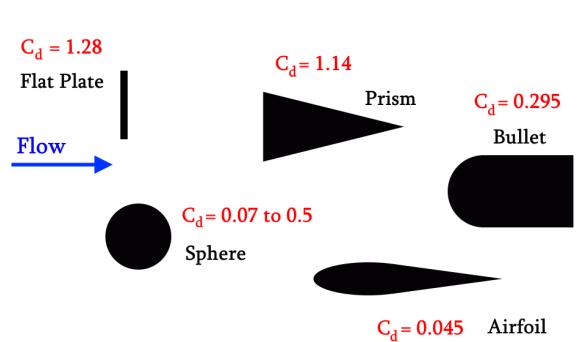


Figure 2.25: Shape effects on drag with same frontal area, figure inspired by NASA [5]

Since these forces generally apply to all objects moving through fluid, this would for a quadcopter not only apply to the propeller's cross section but also the rest of it, such as the geometry of the airfoil or the arm which the motors are mounted in.

The performance of cross sectional shapes have been researched and quantized with a drag coefficient [5], which is worth considering during the design process of a quadcopter. As seen in figure 2.25, trivial additions such as a fillet or reducing the cross sectional area can drastically reduce the drag, making the drone more agile. Downwash is a term referring to how the fluid gets pushed down from the aerofoil. Study of the airflow can be an important property, as it can drastically impact the performance of the quadcopter. A phenomenon called the grounding effect occurs when moving close to a surface, as the air gets pushed back from the surface and provides higher pressure under the aerofoil, thus generating more lift.

## 2.18 Drone Pendulum Fallacy

It may seem intuitive that lowering a quadcopter's center of gravity (CoG) would increase stability by applying a self-righting force when tilted, but since airborne bodies rotate around their own mass center instead of the center of thrust this isn't the case. Consider the simplified case of lateral movement a quadcopter with rotation about either roll  $\phi$  or pitch  $\theta$ . This scenario is illustrated in figure 2.26, where the moment generated by the propellers thrust cancel each other out, as the moment arm is equal due to symmetry. As the force due to gravity coincides with the pivot point, CoG, this does not generate any moment neither. Keeping the center of gravity closer to the plane formed by the four propellers reduces the moment of inertia, thus making the quadcopter more agile as it is less resistant to angular velocity.

Furthermore, a quadcopter will experience some degree of drag on the propellers, where this placement is also beneficial to prevent moment generated by these forces. Vertical offset from the plane will impact the movement of the quadcopter, where a high CoG can make it less impacted by external disturbances, whilst a low CoG can make it more stable during lateral flight [16]. This is shown in figures 2.27 and 2.28.

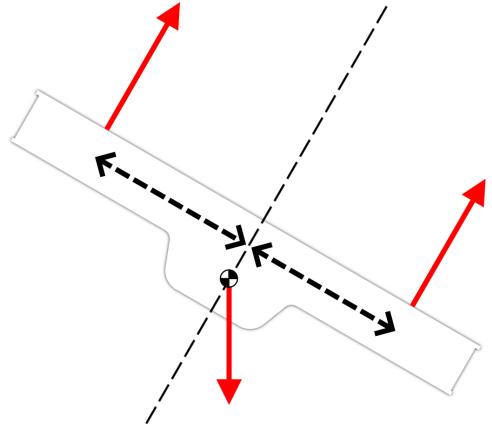


Figure 2.26: Center of gravity acting as pivot point for thrust

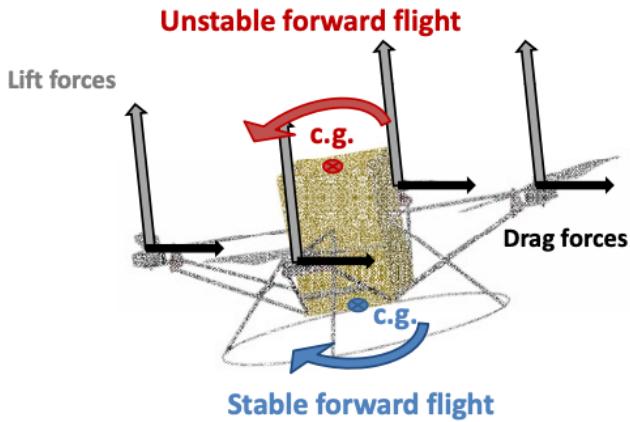


Figure 2.27: Impact of CoG placement during lateral flight [16]

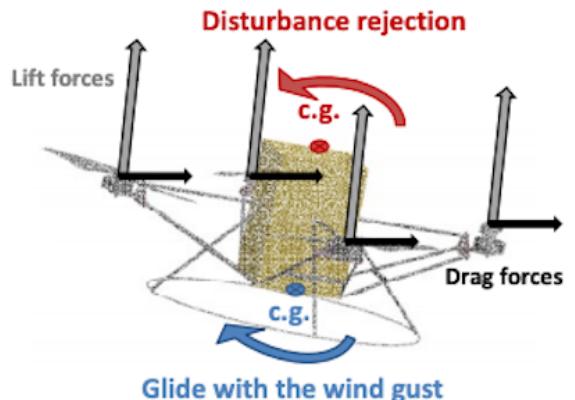


Figure 2.28: Impact of CoG placement on external disturbances [16]

# 3 Concept Development

## 3.1 Overview and requirements

The design this project is based on is the AI oriented Drone Reinforcement Learning Arena from 2022 [14]. It is the latest report from this lineup of drone-based bachelor theses, dating back to the 3DOF Quadcopter platform in 2017 [49]. With a multitude of different electrical, mechanical and software designs, the evolution has provided valuable insight for further development. The objective of this year's project will be to prepare the arena for Reinforcement Learning. Therefore it is crucial to establish some requirements for the system and restrict the scope adequately. As the problem statement is described in section 1.3, this chapter will focus on the overview of the projects overall structure. The concepts were designed with the modular design in mind, allowing for expansion outside the arena. This is further emphasised later in the report, regarding the visual system's role in the state estimator. The concepts directly affects the physical system as well, since the component choices are based on them.

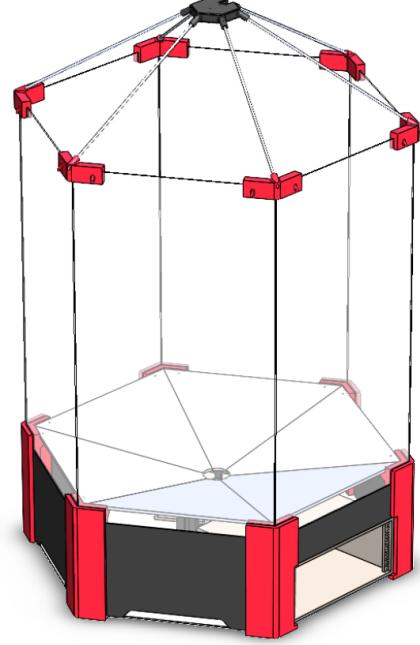


Figure 3.1: Visual Representation of the Drone arena from the 2022 bachelor's thesis [14]

## 3.2 Concept Ideas

The working method of the drone arena system will decide how the rest of the components are implemented, such as how a drone flight iteration would look like, how data from the sensors is handled, how the drone is driven after the AI has processed the data, etc. Therefore it is important to discuss and reflect the various concepts one may come across, in order to have a good foundation to work on. The concepts are somewhat restricted since this years project will be continue on the existing development of the arena, but it is still important to discuss how the system should be structured. To facilitate the decision making process in the concept phase, a Pugh Matrix provides a good overview. Each concept gets either +1, -1 or 0 points for each criterion, which is then weighted to account for its significance. The total sum of each concept gives a final score for each of them, simplifying the choice. A positive score reflects the strength for most criteria, but for the complexity criteria a positive score means that it is less complex, therefore a favorable attribute.

### **3.2.1 Concept 1: Arena Centered**

This concept relies on robust algorithms for the arena system, as it will do all of the calculations in the arena, as well as only have sensors mounted in its environment. This will allow for simpler implementation, as everything is done in the same place. It will still communicate with the drone, but only for sending the values from the RL model to the UAVs actuators. The arena has greater capability compared to the drone, allowing for more processing. The centralization on the arena could give the benefit of increased battery longevity for the drone and therefore longer flight time.

### **3.2.2 Concept 2: Drone Centered**

This solution takes the opposite approach, having everything in the drone. This has the same benefit of having mostly everything in the same place. Having sensors exclusively on the drone allows for scalability in the sense that the drone may be further developed for autonomous flight outside the arena. However, it would for this project still need communication with the arena since the actuators in the middle and the chargers are a vital part of the current system. This solution takes a different approach regarding the data, as it would present it raw to the RL model to save space and computation time, or need a more capable compute module. For some models, raw data could be considered an adequate representation.

### **3.2.3 Concept 3: Hybrid Solution**

This hybrid solution combines the two aforementioned suggestions, providing sensors and actuators on both the arena and drone. This solution may be a better approach as it provides more sensor data, which could lead to a greater state estimate. It will have the arena do most of the heavy computations, with the drones software mainly serving as an interface between the data from the arena and the data to the actuators as well as from the sensors on itself. This implies the vital communication between the drone and the arena for data transfer. A modular design with a solid framework could support expansion outside of the arena, as well as interoperability with other components, or even different drones.

### 3.3 Concept Evaluation

The decision was made with the projects scope in mind, which is to prepare for reinforcement learning with the arena for educational and demonstration purposes, as well as scalability outside the arena. The three concepts presented in section 3.2 have been taken into consideration, where their advantages and disadvantages are demonstrated in table 3.1 and will be discussed in this section.

Table 3.1: Pugh Matrix

Criteria	Solutions/Ideas			Weighting
	Arena Centered	Drone Centered	Hybrid	
Deployability of drone outside arena	-1	1	1	4
Software Complexity	1	0	-1	2
Hardware Complexity	1	-1	0	2
Expected State Estimate Accuracy	-1	0	1	5
RL Compatibility	1	-1	1	3
Quantity of positives	3	1	3	
Quantity of zeroes	0	2	1	
Quantity of negatives	2	2	1	
Total Weighted Score	-2	-1	10	

#### Deployability

Deploying the drone outside of the arena has been taken into consideration, as the university envisioned autonomous flight in an outside environment in the future. Using only built-in sensors on the arena for data gathering limits deployability, as opposed to using sensors on the drone to make it aware of its surroundings. The hybrid solution would also make it possible for such expandability by having a modular software, where the arenas sensors could be replaced with external data providers like GPS or an altimeter and still be used for the rest of the system. This modular approach could also support interoperability, if for instance one desired to swap the actual drone inside the arena. A modular design may be achievable through the use of networked systems and containerization.

#### Software Complexity and Accuracy

The simplest software solution of having no sensors on the drone could be quicker to implement, but may not provide as good data compared to the other solutions. This is illustrated in table 3.1, where "Software Complexity" has an inverse relationship to "Expected State Estimate Accuracy". The drone centered solution received a neutral complexity score since it would be simpler than the hybrid solution, as it has everything integrated in the drone but still needs connection with the arena. Since the simplicity is less important than the performance, the hybrid combination would get the best overall complexity/accuracy results. This is reflected by the fact that the group desires the best possible state estimate. The expected accuracy of the estimator for the hybrid solution came out best due to the fact that it has more sensors available. Though the drone centered concept may be subject to more sensors when expanding outside the arena, the primary focus is a design inside the arena. Inside the arena it has less support for more sensors like an altimeter/barometer or GPS (as it is inside), not to mention the fact that it could be challenging to fit more sensors on the drone. Furthermore, a hybrid solution could still be achievable with wireless communication even if the drone is outside the arena.

## **Hardware Complexity**

Hardware is also an essential aspect of the concepts, where a robust yet feasible design is desirable. Therefore the drone centralized concept was rated lowest, due to the complexity involved in a complete flying system with everything integrated. Both physical dimensions and the fragility of the more complex components would be limiting factors for this design. The hybrid solution got a neutral score as it would have simple sensors in the drone which are easy to protect and does not take up a lot of space, whilst the arena centered approach was rated with a positive score due to even less complexity.

## **Reinforcement Learning Compatibility**

The RL compatibility criterion was added to quantify how easy the solution would be to use for reinforcement learning. As the project is for educational purposes, the solutions providing vector representation would be desirable as they require less complex RL models. Having everything on board will most likely be a limitation, which will lead to a restricted model which could have a harder time interpreting the raw data.

## **Final Concept**

Given the results from the Pugh matrix shown in table 3.1, the hybrid solution came out on top. Despite its significantly higher score, the criteria showcased the advantages and disadvantages of the other solutions as well. The key insight which was gained from the discussion was the deployability of the drone outside the arena. Initially the drone centered concept seemed like the obvious choice, as it could be easily used for autonomous flight outside the arena with different algorithms. Further discussion led to the conclusion that a modular software design could support both deployment outside the arena as well as interoperability inside the arena. Thus a choice was made with the hybrid solution, combining the best from all solutions with a modular design in mind. Overall, the Pugh matrix gave a good overview and made the concept choice more efficient.

# 4 Design and implementation

## 4.1 System Architecture and Design

The architectural structure of the system is illustrated in figure 4.1. This shows a simplified birds eye view of the pathways of data transfer between the components of the whole system.

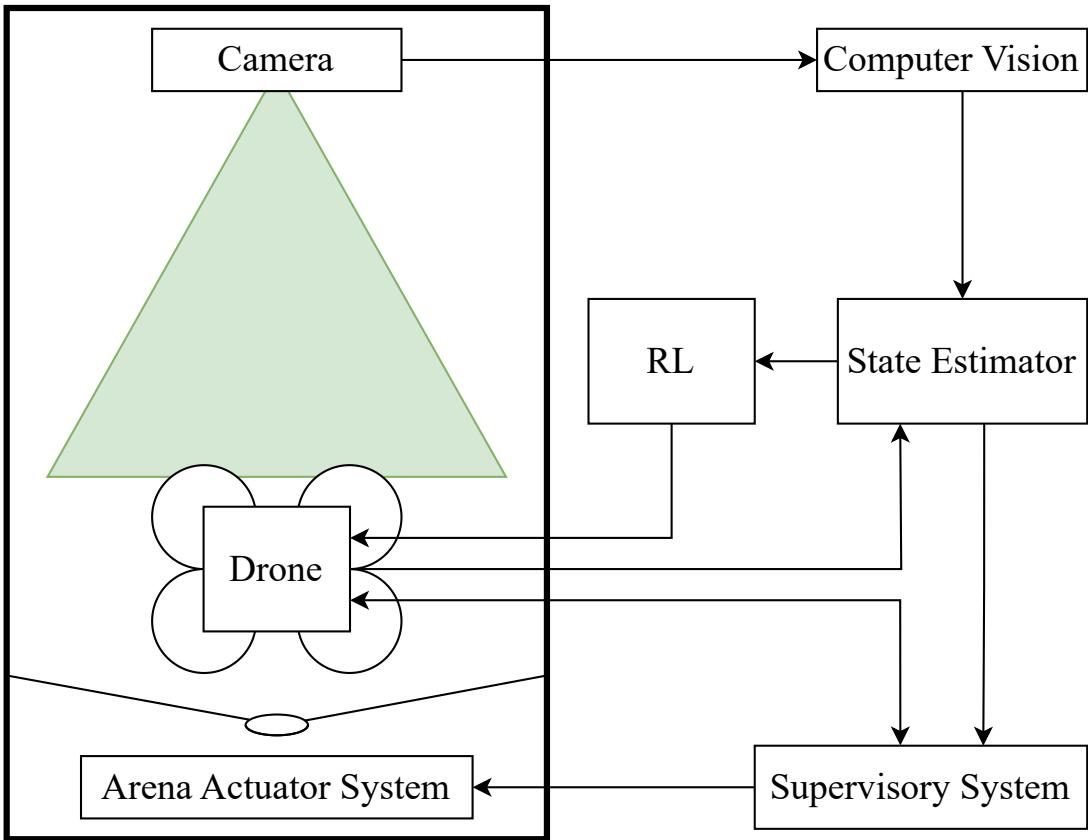


Figure 4.1: System architecture block diagram

The envisioned future of the project is to estimate the state of the drone, where the machine vision system was chosen as the low rate "sensor" module for the filter. The chosen machine vision algorithm utilizes fiducial markers, given their suitable advantages. The camera used for this project was the Intel RealSense D435, as it was available for prototyping immediately since the RL arena group from 2022 evaluated the choice and provided it for use [14]. As the pose estimation from the fiducial markers provide more information of the drone, its development was prioritized over the depth estimate from the stereo vision. Throughout the project, the primary formalisms used to represent the orientation of the three dimensional rigid body are the rotation matrices and rotation vectors.

## 4.2 Project Workflow

### 4.2.1 Version Control

Version control has been an integral part throughout the development of the entire project. For this, the group has used git and the online platform GitHub to both push changes for each other and also keep backups for rolling back, in case of undesired changes. ".gitignore" files were added to build files so that they would not be pushed. It also enables simple portability between the machines, working between Windows and Linux. To use git on Windows, git bash has been used to combat its limitations compared to the terminal on Linux. The repository is hosted online with GitHub, and has been divided into directories to keep the version control organized. The repository architecture is explained in further detail in the "README.md" file located in the main directory.

### 4.2.2 Project Management

Throughout the project, two platforms have been used other than version control on GitHub; namely discord and google drive. The former is a less formal platform, which has been primarily used for communication regarding technical discussions and questions. Google drive has been used mostly for planning, such as Google docs for weekly meetings to keep the groups members up to date and ensure that the project was developing at expected speeds. In order to keep track of the projects progress, a Gantt chart was created in Google sheets with a couple functions, inspired by a tutorial from "Vertex42" on YouTube [129]. It allowed a visual overview of the project's tasks and how much time should be allocated for each task to meet the deadlines and expectations. It was also used for planning so important dates were noted and taken into account, such as exams parallel to the bachelor project. The final version of the Gantt chart is shown in appendix D.1. A Kanban board was also utilized to track the specific tasks, depending on the available resources. This was done using the online platform "Miro", shown in appendix D.2.

## 4.3 Linux Computer Setup

### 4.3.1 Desktop Setup

A desktop was set up with Ubuntu with LTS version 22.04 Jammy Jellyfish for development. A user with sudo privileges was also made for every member of the group. ROS2 Humble was also installed by following the installation guide for binary debian packages on the ROS2 humble docs [101]. The dependencies for all users from `apt` are

Table 4.1: Dependencies: pip

Dependency	Version
numpy	1.26.4
opencv-contrib-python	4.9.0.80
opencv-python	4.9.0.80
pandas	2.2.2
protobuf	3.12.4
pyperf	2.6.3
pyrealsense2	2.54.2.5684
python-dateutil	2.9.0.post0
qtm	2.1.2
qtm-rt	3.0.1
six	1.16.0
tzdata	2024.1

Table 4.2: apt

Dependency
usbview
valgrind
net-tools
nmap
pkg-config
gdb
protobuf-compiler
python3-pip
make

Table 4.3: apt

Dependency
librsclibrationtool
libudev-dev
libssl-dev
libusb-1.0-0-dev
gh
git
binutils-avr
build-essential
cmake
ffmpeg

### 4.3.2 Secure Shell and X11 Forwarding

To enable connection with the computer with its dependencies and connected devices, Secure Shell (SSH) and X11 were used. SSH ensures a secure connection with encryption, while X11 enables the use of a display to complement the terminal. Though not great for streaming due to limited framerate and the limited ability to physically move the objects in the frame, it enabled software development and logic checks. Most important, it allowed work outside of campus provided the use of a vpn (Virtual Private Network) to connect to the UiA network. This was achieved through the eduVPN client.



Figure 4.2: Xming  
Logo [17]

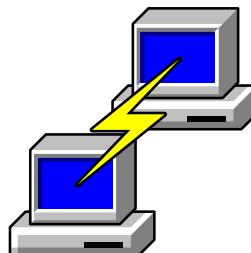


Figure 4.3: PuTTY  
Logo [112]



Figure 4.4: eduVPN  
Client Logo [10]

For X11-forwarding to windows, the program PuTTY and the X servers VcXsrv and Xming had to be used on the windows machine, where X11 forwarding was enabled in PuTTY. This was done in PuTTY by checking the "Enable X11 forwarding" setting as well as entering the host name and IP (in the form `username@xxx.xxx.xxx.xxx`) together with setting the port. SSH without X11-forwarding could also be used through the windows terminal with openSSH installed through the optional features page in windows settings.

Vim along with VScode with the Remote Explorer file editor plugin was primarily used for any text editing through ssh, mostly used for programming on the raspberry pi, to increase workflow,

suckless-tools was installed so that the tabbed feature could be used to implement tabs in xterm using:

```
$ tabbed -c xterm -into &
```

Using this method, tabs could be created with shift + ctrl + enter and ctrl + 1,2,3,... to switch to the tab. `~/.vimrc` and `~/.Xdefaults` were created and modified to edit the functionality of vim and the visual look of xterm for more comfortable usage.

#### 4.3.3 Arena Raspberry PI setup

The arena components are controlled by a Raspberry PI running ROS2. Ubuntu server LTS 22.04 was installed on a raspberry pi 4B using the Raspberry PI Imager tool. Ros2-humble-desktop was then installed following the same method as on the main computer.

The raspberry pi was setup with a static ip so that the group did not have to connect a monitor and keyboard to see the dynamically allocated ip which changed often. The following contains information about the UIA network, this will be different on a separate network. First the subnetmask was found with ifconfig:

```
$ ifconfig | grep netmask
```

Which was found to be 255.255.254.0 which corresponds to the CIDR code of /23 using a CIDR lookup table, the ip range was then found using ip route [36]:

```
$ ip route | grep -E "default via"
```

This along with the netmask shows that there was an ip range of 128.39.202.0/23 in the local network, which corresponds to the ip ranges of 128.39.202.0-128.39.203.255. Using this information, an unused ip address was found using nmap:

```
$ nmap -sP -v 128.39.202.0/23 | grep down
```

After choosing an unused ip address, the universities DNS nameserver was found, the easiest way to find this is to use another computers's graphical interface for internet connections to see the DNS addresses. A file in `/etc/netplan` was created named `01_config.yaml`, the first two numbers decide the order of which file in the directory will be used to setup the network, and the rest of the name is irrelevant, the file `/etc/netplan/01_config.yaml` contained the following:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      addresses: [128.39.202.209/23]
      routes:
        - to: default
          via: 128.39.202.1
      nameservers:
        addresses: [158.37.218.20,158.37.218.21]
```

#### 4.4 Intel Realsense D435 Setup

OpenCV has been used together with the Python wrapper for the Intel Realsense D435 camera, namely their libraries `cv2` (OpenCV) and `pyrealsense2` (RealSense Wrapper). The `numpy` library has also been utilized often for mathematical expressions such as finding the length of a vector or for working with arrays, such as setting up the camera matrix or distortion coefficients. The python

library `os` assured the correct file path for import/export. Furthermore, the `glob` library has also been used for importing multiple files in a directory.

#### 4.4.1 Camera Information - Software Development Kit Setup

In order to find information (such as the distortion model and accepted formats) about the camera, the group contacted Intel RealSense Help Center [123]. The following command was suggested to write in the terminal:

```
$ rs-enumerate-devices -c
```

This initially resulted with the terminal not recognizing the command. In order to use the command, RealSense SDK 2.0 had to be built. This was done by following the official build guide for Linux Ubuntu installation, with a few modifications [26]. Due to these modifications, the details of how the guide was followed will be elaborated in the rest of this section.

To prevent conflicts with other parts of the project, the operating system was not updated since it was already running a LTS version with a stable kernel. Since this was done only for extracting information of the camera from the terminal, an IDE was not installed. Therefore, the "**Install dependencies**" part of the guide was only done from step 2 to 4.

The next step was to "**Install librealsense2**", which was done by cloning their GitHub repository:

```
$ git clone https://github.com/IntelRealSense/librealsense.git
```

Inside the cloned repository (now referred to as the `librealsense2` root directory), the permissions script had to be run in sudo mode with enhanced privileges with the following command:

```
$ sudo ./scripts/setup_udev_rules.sh
```

In order to build and apply the patched kernel modules for the LTS version used at the time, the following command was also run with sudo for higher privileges:

```
$ sudo ./scripts/patch-realsense-ubuntu-lts-hwe.sh
```

With `librealsense2` installed, it was time for "**Building librealsense2 SDK**". Inside the same root directory as the previous step, another directory was created and entered:

```
$ mkdir build && cd build
```

Making use of the newly created directory, the `cmake` configure step was done in its default state:

```
$ cmake ../
```

With the Software Development Kit finished building, the initially suggested command was ultimately ran in the terminal without issues and the results from the terminal documented.

#### 4.4.2 Camera Readings

To establish a foundation to work from, a live reading of the camera was initially a simple copy of Nick DiFilippos source code [34], shown in G.1.1. It was done in order to assure functional hardware and software. The camera was set up to stream video with the `pyrealsense2` wrapper, using pipelines. The configuration for both the depth and color stream was initially set up with a resolution of 640x480 pixels and 30 fps, as well as the formats for the color and depth streams respectively being `bgr8` and `z16`. The connection was set up using an unidentified cable from USB type A to USB type C. There were issues with achieving a higher resolution, which was fixed by

changing the data cable to a RS Pro USB 3.1 cable [21] provided by the faculty. This file served as a baseline for experimentation of the configurations for the camera streams. If sentences have been used here as well as in later programs in order to control the program while running for things like capturing the current frame as an image file or breaking the loop.

In order to document the different formats for the camera modules, a different script was created to take screenshots of the current frame from the stream. The screenshots were then labeled with an incrementing number at the end, relative to the amount of items inside the folder. This was done with a suggestion from the language model "ChatGPT" to use the `listdir()` method from the `os` library, as well as finding the length of this list. Each files content was commented in the `README.md` file in the corresponding directory. The source code is shown in appendix G.1.2. The program was set up so pressing "s" on the keyboard saves the current frame, whilst "q" stops the stream. Later it was also expanded so pressing "c" saves the frame to a different directory, for separating general screenshots and frames captured for calibration. The different configurations for the depth, color and infrared streams were tested and documented individually. To remap the depth stream for display (visual for inspection), the `applyColorMap()` method from OpenCV was used. It takes in the source (current frame) and the colormap to apply. The source was then scaled to fit the colormap to the desired range with the `convertScaleAbs()` method from OpenCV using a scaling factor `alpha`. A highlight of the relevant screenshot code is shown in listing 4.1.

Listing 4.1: Highlight from `screenshotCamera.py`

```
# Add color coding if using depth stream
image = cv2.applyColorMap(cv2.convertScaleAbs(image, alpha=0.3), ...
    cv2.COLORMAP_TURBO)
```

Various colormaps were explored with `COLORMAP_TURBO` being the most appropriate, providing the most dynamic depth display. This color scale is shown in figure 4.5.



Figure 4.5: `cv2.COLORMAP_TURBO` scale from OpenCV documentation [84]

## 4.5 Machine Vision Development - Outside of Arena

### 4.5.1 Marker Detection

To get familiar with marker detection with OpenCV and Python, a simple implementation of marker detection was done. This utilized the aruco-module from the openCV library and the `pyrealsense2` wrapper for establishing a live stream with the RGB module of the D435 camera. The stream and approach for screenshots were set up in a similar fashion as subsection 4.4.2, expanded with the marker tracking. In order to identify the marker, the image was turned to grayscale using the `cvtColor()` method and the conversion constant `COLOR_BGR2GRAY` from the OpenCV library. Subsequently, the aruco-modules method `detectMarkers()` was used on this grayscale image with the arguments for the image, setting up which dictionary to use and the parameters from the `DetectorParameters` class. Initially the values from the marker were only printed in the terminal during development, further expanding to the actual image. The library's method `drawDetectedMarkers()` was used to illustrate the edges of the marker on the colored image with the corner coordinates from the grayscale image. Furthermore, a code segment was implemented to display the marker ID on the colored image; heavily inspired by `pyImageSearch` [98]. The full source code is shown in appendix G.1.3.

#### 4.5.2 Intrinsic Parameters

When the streaming and marker detection was working correctly, the next objective was to obtain a pose estimate using the libraries available. The methods available with the library utilizes both the camera matrix and distortion coefficients, which meant extraction of the cameras intrinsics was in order. Initially calibration methods were explored, prior to the supervisors suggestion of extracting the existing intrinsics from the camera through its related software. The latter option was done using CustomRW, which was installed with the D400 Dynamic Calibration Tool using the Debian packages from the Intel server with the terminal by following their official guide [120]. After installation, the calibration was factory reset in order to remove using the following command in the Linux terminal:

```
$ Intel.RealSense.CustomRW -g
```

Where the `-g` flag corresponds to the gold factory calibration settings. The intrinsic parameters were then read using the `-r` flag:

```
$ Intel.RealSense.CustomRW -r
```

It was only later realized that the camera intrinsics could also have been extracted from the SDK solution described in 4.4.1.

#### 4.5.3 Marker Pose Estimate

At first, a pose estimate from the markers was set up in its own script, shown in G.1.4. The previous methods for streaming and marker detection were expanded with the pose estimation. The actual pose estimate was set up by checking for corners in the current frame, then iterating through all the markers in a for loop (mostly just one, but with the ability to add more, for example on the top and bottom of the drone). Inside this the essential method `estimatePoseSingleMarkers()` was used for finding the translation vector representing the placement of the marker and the rotation vector for its attitude. The previously found intrinsics (namely the camera matrix and distortion coefficients) were needed as arguments for this method, as well as the detected corners and the real measured size of the fiducial marker. Further the `drawFrameAxes()` method was utilised to display the axes on the marker in the image for visual feedback of the estimate. It was later discovered that these methods were deprecated and replaced with other methods. As the successors require more time for setting up and the prior was already configured and working correctly (and still supported with the 4.6.0 version of OpenCV's documentation [87]), they were set aside for reconsideration in the future if issues arose with the current program.

#### 4.5.4 Camera Calibration and Translation Measurement Test

From this point onwards, the RGB camera was used with 848x480 pixels (480p) to work with 60 fps. With the state estimate giving an output, the translation vector's distances were compared with the measured values using a ruler for validation<sup>1</sup>. Two primary issues arose; the marker was difficult to read at longer distances and the fact that the difference between the measured and calculated value increased proportionally with the real distance. The first issue was addressed with the option for a larger physical marker and a reduced dictionary. Furthermore, reducing the resolution of the RGB camera for higher frame rate resulted with even more difficulty for tracking. The intrinsics also had to be updated when changing resolutions, by reading the terminal using the software development kit documented in section 4.4.1. The incorrect scale was corrected by calibrating the camera, since the gold factory calibration settings proved inefficient. The first calibration was opened by running the following terminal command from the D400 Dynamic Calibration Tool

```
$ /usr/bin/Intel.RealSense.DynamicCalibrator
```

<sup>1</sup>This was done outside of the arena. Actual testing in the arena will be discussed later in the report.

After doing the calibration with the Dynamic Tool, the parameters were checked with the SDK enumeration command. This also gave unsatisfying results when implemented into the pose estimation script, so a calibration was done with OpenCV in its own script, shown in appendix G.1.5.

The OpenCV calibration was done with inspiration from Nicolai Nielsen's tutorial from YouTube [75] and OpenCV's documentation [82]. The images to do the calibration on were imported from the screenshotCamera script in the calibrationCaps folder using the `glob` library. A custom chessboard was printed out, with square sizes of 19 millimeters and 13x9 vertices. The script was set to iterate through all the imported images. Despite dr. Zhang's paper stating only 3 captured images necessary [133], OpenCV explicitly states it needs at least 10 images [82]. As the `calibrateCamera()` method requires both the 2-dimensional image points and the 3-dimensional image points, some setup was needed. The 3D world points were set up with the coordinate frame planted on the printed chessboard, which then was set up as a template array. This array was then appended to a dynamically expanding list for each time there was identified corresponding 2D image points. OpenCV's method `findChessBoardCorners()` was used for finding the corners in the imported images. Before sending the corner coordinates into the 2D array, a more accurate estimate of their location was implemented through the `cornerSubPix()` method. It was set up to do the algorithm on the grayscale image with the corners from `findChessboardCorners()`, as well as the criteria and the `winSize` and `zeroZone` requirement. The search window size was set to  $11 \times 11$ , the singularity protection was disabled by setting `zeroZone = (-1, -1)` and the criteria was set to maximum iterate 30 times and have a tolerance of 0.001 pixels. After the sub-pixel algorithm was implemented for the corner coordinates, the two expanding arrays were set up as input arguments for the calibration method along with the images resolution in pixels.

To visually verify the calibration, the corner detection was displayed on the current frame with the `drawChessboardCorners()` function on the colored image. After the calibration, the colored image was displayed both with and without distortion using the `undistort()` method. They were set up with a similar fashion as the screenshot script, to export the resulting images to the Results folder.

After the calibration was complete, the manual measurement test was completed. It was done by comparing the third component of the translation vector to the height between the camera and marker, measured with a ruler. The x and y directions were also measured and compared to the respective elements of the translation vector.

#### 4.5.5 Depth Readings

The depth module readings were done outside of the arena, comparing to measured distances with a ruler like before. Prior to any readings of the depth through source code, the Intel RealSense Viewer was used to perform a health check [25]. This was done by running the "On-Chip Calibration" under the more tab while having the camera stationary pointed at a wall. As the health-check was satisfactory, the group proceeded with the development of depth readings through a python script.

The python script was set up like before with pipeline for streaming, as well as a loop to display the colormapped image and the option to pause or stop the stream. After this, the script was set up to simply read the depth at the pixel coordinates in the middle of the image. The script was also expanded with an additional function for reading the average value of the past 10 frames' depth values when pressing pause. This made the documentation of the read values at each distance easier, as the values varied a fair amount. The script is shown in appendix G.1.6.

#### 4.5.6 Combined Depth and Marker Estimate

After the pose reading and depth reading scripts were set up individually, both were integrated in a single script, shown in appendix G.1.7. This was done by modifying the arucoPoseEstimation script and expanding it with the depth reading. First, the code was simplified as the ID display on the

screen was not needed. The final script combining both has the color and depth streams set up like before, as well as the marker detection and pose estimate. Both the pose and depth was done inside a for loop (inside an if-sentence), iterating through the detected markers if the corners are detected in the current frame. The depth reading was implemented by calculating the pixel coordinate of the middle of the marker and extracting the distance at this coordinate. The middle coordinate was found by calculating the average of the corner coordinates, both for x and y. In order to do this the corners were extracted like before for displaying the ID on the image, with NumPy's `reshape()` method for presenting them as a  $4 \times 2$  array with the x and y coordinates in the two columns. The crosshair used in the previous script was implemented to track this coordinate, first in the color image to confirm the correct location; then in the depth image to keep track of the marker in both streams.

## 4.6 Machine Vision Development - inside the Arena

### 4.6.1 Marker Dictionary Test

In order determine the best suitable marker, a test was completed to demonstrate the performance of the different dictionaries inside the arena. The test was done by recording a video with the camera and saving it as a file to be read later. The video was recorded while capturing multiple markers in order to retain the same movement of the camera, enabling a fair comparison. The frames were recorded with `pyrealsense2` as before, then saved using the MJPG (Motion Joint Photographic Experts Group) format inside an AVI (Audio Video Interleave) container, through OpenCVs `VideoWriter()` method. The recordings were then saved to a local directory, to prevent large files in version control. This recording script is shown in appendix G.1.8. After the recording was done, the video was then read and analyzed in another script shown in appendix G.1.9. It was done by making a list of the dictionaries and iterating through the list, playing the video back for each dictionary with OpenCVs `VideoCapture()` and `read()` methods. A pose estimate was done for each dictionary like before inside the "playback" while-loop, which was nested inside the "iterating dictionary" for-loop. Initially the analyzer script was done with exporting values to a csv (comma-separated values) file for further analysis, which proved redundant as all calculations were done in the same script; hence why the csv-addition was removed later. Another script was also created for visual inspection by exporting the recording, one video for each dictionary with pose estimate. This is shown in appendix G.1.10

With the playback and pose estimation set up, it was time for determining which dictionary was best suited for the project. Three metrics were initially measured to accomplish this; computation time, number of frames with data and false positives. The number of false positives was approximated by finding the magnitude of the translation vector, using the `linalg.norm()` function from the `numpy` library, and comparing the difference between the current and last frames magnitude to a constant tolerance. This method was tested, but it proved simpler to visually inspect it through the exported videos, especially since it was not a common issue with false positives. The computation time was documented by taking the difference between when the current dictionary started and stopped. Finally, the number of frames with data was measured to quantify how often the marker was detected, by incrementing a variable every time it was detected in the loop. A similar variable was set up to increment so the total amount of frames were also documented. These metrics are only indicative, thus visual inspection of the recording with the display of each dictionary pose estimate proved to be most efficient and reliable.

### 4.6.2 Revised Machine Vision Algorithm

When the physical setup was moved to the arena, some issues were discovered. The pose estimate method did not have a unique solution, so the orientation turned unstable. An optical illusion gave the impression that it was turned upside-down, when it really was rotated a smaller degree, due to the axes being displayed on a 2D screen without a sense of depth. This ambiguity was proposed

to be solved with a multitude of solutions. The code was modified to be used with variants of OpenCV's "Perspective-n-Point" functions, with primary focus on `solvePnPGeneric()` as it had the option to extract all possible solutions per frame. When enabling the `useExtrinsicGuess` parameter and the `SOLVEPNP_ITERATIVE` flag, the function took in a rotation and translation vector as initial conditions per frame and based the choice of the next vectors on the previous vectors. This combination requires an initial vector to compare the previous to, so this was enabled after a couple of detected frames. This solution is shown in listing 4.2, but was not kept for the final version of the program.

Listing 4.2: Highlight from previous iteration of poseMarkerDepth.py

```
# SolvePnPGeneric for extracting all possible vectors
if (roundNr < 1):
    retVal, rotVectors, transVectors, reprojError = cv2.solvePnPGeneric(
        markerPoints, markerCorner, cameraMatrix, distortionCoefficients, ...
        rvecs=rotVectors, tvecs=transVectors, ...
        reprojectionError=reprojError)
else:
    print("\n----- 1 has passed -----")
    retVal, rotVectors, transVectors, reprojError = cv2.solvePnPGeneric(
        markerPoints, markerCorner, cameraMatrix, distortionCoefficients, ...
        rvecs=rotVectors, tvecs=transVectors, ...
        reprojectionError=reprojError,
        useExtrinsicGuess=True, flags=cv2.SOLVEPNP_ITERATIVE, ...
        rvec=rotVector, tvec=transVector)
```

It makes the axes significantly more stable, but the pose may still be incorrect, as certain attitudes may lead the algorithm to choose the wrong vector and "latch onto" the incorrect pose. Next `useExtrinsicGuess` was turned off and the flag was replaced by others which enabled the extraction of multiple solutions per attitude. The angle between the previous vector and both current solutions were compared and the choice based on this, with same behaviour as the previous combination. This was done as follows:

Listing 4.3: Comparison and logic from another iteration of poseMarkerDepth.py

```
if len(rotVectors) > 1:
    # Extract previous vector (SINGLE)
    prevRotMat, _ = cv2.Rodrigues(prevRotVector)

    # delta Rotation 0
    rotMat0, _ = cv2.Rodrigues(rotVectors[0])
    rotMatDiff0 = numpy.dot(prevRotMat, rotMat0.T) # previously rotMatDiff
    rotVecDiff0, _ = cv2.Rodrigues(rotMatDiff0)
    # delta Rotation 1
    rotMat1, _ = cv2.Rodrigues(rotVectors[1])
    rotMatDiff1 = numpy.dot(prevRotMat, rotMat1.T) # previously rotMatDiff
    rotVecDiff1, _ = cv2.Rodrigues(rotMatDiff1)

    # Angles between current solutions and previous vector
    angleError0 = numpy.linalg.norm(rotVecDiff0)
    angleError1 = numpy.linalg.norm(rotVecDiff1)

    if angleError0 < angleError1:
        rotVector = rotVectors[0]
        print("\nTwo solutions, chose 0")
    else:
        rotVector = rotVectors[1]
        print("\nTwo solutions, chose 1")
```

As shown above, no simple logic would solve the ambiguity issue without further information. A couple options were proposed to serve additional information to provide a unique solution. Fusion

of the depth camera may give enough information to give an indication of the orientation, as the depth values at each corner are available so it may be constrained close to the plane these corners lay in. Similarly, the light levels may be possible to use to indicate which part of the marker is closest to the camera. Alternatively more markers could have been utilized, either coplanar or not (such as a cube), giving more information of the image as we know the physical relations between them, restraining the possible orientations. Furthermore, other algorithms could also have been explored which gives a pose estimate with a different algorithm and other parameters (such as not only the four points but also the marker's binary code itself). The final option, which was chosen to continue with, was to export both vector solutions and determine the best fitting outside of the current script, based on additional information from the IMU. The extraction of both the vector solutions was done by disabling the `useExtrinsicGuess` and setting the `SOLVEPNP_IPPE` flag. Since the previous center coordinate used for depth reading was implemented incorrectly, the script was planned to export the raw color and depth frames for the RL model if desired. Furthermore, the algorithm for subpixel coordinates of the corners was briefly implemented, but yielded unstable results so was shortly removed after. This script was used further as the main machine vision script for communication with the rest of the system and RL model. The final version, with the communication included, is shown in G.1.11.

## 4.7 Pose Validation with Qualisys

### 4.7.1 Pose Validation Test Concept

In order to validate the estimate of the pose from the markers and depth, a test was designed. The test was made for quantifying the performance of the arena's machine vision system by comparing the measurements to a reference value, with a design such that the tests are reproducible. The premise of the test was to film the scene with the D435 camera mounted in the arena, together with the motion-capture cameras from UiA's Qualisys system. The movement could then be recorded and compared post-processing to verify the fiducial marker pose estimate around the scene, as the motion-capture system is calibrated and known for giving a precise estimate, thus providing reference characteristics. The recording would capture the marker movement when moved around with a slender rod to prevent occlusion. The test is not only designed for logging the error, but also for post-processing analysis such as making distributions for the accuracy of the system which is desired for state estimation filters. To keep the systems separate and not let the unit under test (UUT) affect the data, the test is designed so the raw data from both systems are recorded and exported prior to marker detection and analysis. Further details regarding the outputs and design of the test are discussed in section 4.7.2.

### 4.7.2 Pose Validation Test Development

In order to make the test reproducible, the equipment was made together with scripts and a test plan. The test plan gives an introduction of the experiment's motive, as well as a description of the procedure and success criteria, and is shown in appendix E.1. It serves as an instruction for future use, so this report is not needed to complete the test, as well as a checklist so future tests will be completed in their entirety. A quick reading of the test plan is advised prior to reading the subsequent sections.

#### 4.7.2.1 Physical Development

The test requires some equipment to move the marker around the scene without occluding the markers (both ArUco and the reflective balls for motion capture), so the concept was then to develop a cube for attachment of the fiducial marker as well as the reflective spheres for the Qualisys system's pose estimation. The cube was modeled in SolidWorks with sides slightly larger than the marker and holes for the carbon fibre rod for operation. As the motion capture balls have threads, holes for attaching screws were added to the cube. The faculty staff gave a tip to place the holes spread out randomly on the cube to prevent symmetry so it is orientationally unique. Furthermore, it was

advised to have two of them placed in the middle of two opposite sides of the cube, so the 6 DOF rigid body frame is simpler to align with the marker as the center point is known. The screws and the rod were attached to their holes with superglue to prevent them from moving. The technical drawing for the 3D model of the cube is shown in appendix A.1.1.

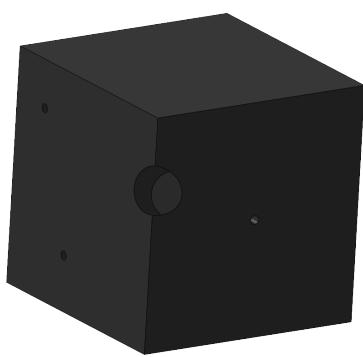


Figure 4.6: 3D Model of Cube

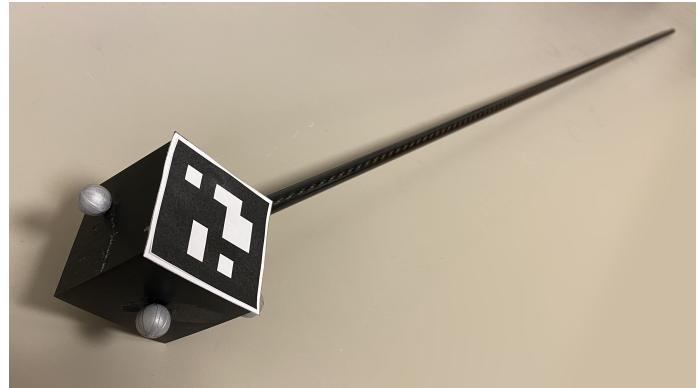


Figure 4.7: Physically assembled tool for pose test

In addition to the test tool shown in figure 4.7, a calibration plate was made to facilitate the setup for the test and further reinforce reproducability. A sketch was made for the design, shown in figure 4.8, which was then used to laser cut an acrylic plate. The sketch was converted to a DXF file for the laser cutting machine, but the three-dimensional technical drawing is shown in appendix A.1.2. It is a circular plate, where the three holes are for placing the Qualisys calibration tool's set screws in so the placement is the same every time. This is shown in figure 4.9.

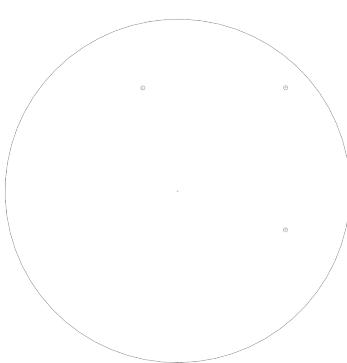


Figure 4.8: Sketch of calibration plate

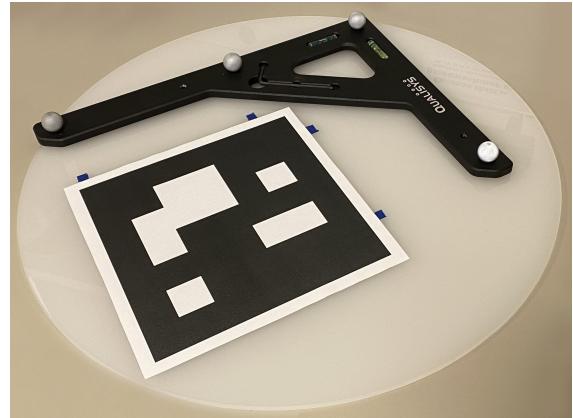


Figure 4.9: Calibration plate

Furthermore, a mount was modeled, 3D printed and glued to the middle of the underside of the calibration plate so it can be rotated around the arena center. The large marker was measured prior to taping it to the plate, ensuring a parallel relation with the Qualisys tool, 50 [mm] from the x-axis and 90 [mm] from the y-axis of the L-frame's inner edge. The large calibration marker was printed with a size of 167 × 167 [mm]. The technical drawing of the actuator bracket is shown in appendix A.1.3.

#### 4.7.2.2 Software Development

To facilitate analysis after using the motion capture setup, the test was designed for exporting raw data. This also enables the possibility for multiple analyses on the same dataset from each test. An overview of the outputs and files used is shown in appendix E.2. As the test exports only raw data, translation and rotation for relating the Qualisys reference frame to the D435 camera frame is done after recording the test. In order to rotate the Qualisys frame to match the orientation of the

camera frame, a 180 degree rotation (or  $\pi$  radians) rotation about the y-axis (pitch) was needed. This transformation can be done with a simple rotation matrix  $R_y(\pi)$  from (2.4):

$$R_y(\pi) = \begin{bmatrix} \cos \pi & 0 & \sin \pi \\ 0 & 1 & 0 \\ -\sin \pi & 0 & \cos \pi \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad (4.1)$$

Whereas the translation between the frames can be done with vector addition. The script made for assisting with aligning the frames orientations was done by comparing the large marker estimated orientation with the rotation matrix from equation (4.1). This was done by calculating the rotation matrix for rotating from the ideal orientation to the measured orientation:

$$R_i^m = (R_c^i)^T \cdot R_c^m \quad (4.2)$$

Where  $R_i^m$  is the difference between the ideal orientation and the measured orientation,  $R_c^i$  is a rotation matrix expressing the ideal orientation from the camera perspective and  $R_c^m$  is the measured rotation matrix. The latter is extracted from the rotation vector given by the state estimate of the large marker on the plate, converted to the matrix form through the `Rodrigues()` method from OpenCV. When the difference matrix  $R_i^m$  is calculated, it is converted back to axis-angle representation, so the angle between the desired orientation and measured orientation is given in degrees by calculating the magnitude.<sup>2</sup> This script is shown in appendix G.2.1. The crosshair display for the D435 camera is a simple script with livestream from the camera and lines in the middle coordinates of the screen, and is shown in appendix G.2.2.

The recording script for the test was set up to record the D435 stream and the internal timestamps of time passed since the script started with the `time.monotonic()` function. This script was set up to name the files created automatically based on the existing files in the destination. This was done to speed up the test process for when multiple recordings were done subsequently without analysis between. This script is shown in G.2.3. The next script for marker detection was set up with a variable so the user can choose which test number to start analysis on. The marker script was set up to export all the data from the marker detection even though some may not be used, as it takes little extra space, and is shown in appendix G.2.4.

The main script for exporting values also imports the files based on a variable with the same name for selecting test number. Since the test exports two separate time series with local start time, the two systems have to be synchronized. As the test which was done for this projects analysis did not have a real time requirement, it was solved by keeping the marker cube still in the beginning and adding a start detection of both time series when the translation vector changes with a set magnitude over a specific amount of frames. The start time of the qualisys data was then subtracted from the time series list. With this the script iterates through the QTM time series and checks each element, where the D435 time of the current frame was subtracted and compared so the scripts compares the data of the frame with the closest timestamp over the current time. This is shown in listing 4.4, where the iterator  $i$  goes through the list until the condition is fulfilled and breaking the loops so it can be used to index the corresponding data. The full script is shown in G.2.5.

---

<sup>2</sup>Still done with the vector norm function from NumPy's linear algebra module: `numpy.linalg.norm()`

Listing 4.4: Highlight from Pose Validation Test Analysis

```
# Match closest timestamps
if detectedStart:
    for i in range(len(timeQTM)):
        if (0 < ((timeQTM[i] - startTimeQTM) - curTime)):
            correctTimeQTM = timeQTM[i]
            break
    print("\nCurrent Time: ", curTime)
    print("\nOpenCV Time: ", timestampOpenCV[loopRound])
    print("\nQTM Time: ", correctTimeQTM)
```

#### 4.7.3 Plotting

This section will cover plotting specific to the completed tests done for this project. The plotting was done with multiple representations to visualize and give a good intuition of the performance for each test, where the full script is shown in appendix G.2.6. The first plot simply compares the position of the translation vector from OpenCV and QTM. The first translation vector was chosen for the plots, based on results from comparing the average absolute difference between the translation vectors. Since the translation between the L-frame and D435 camera was done for all frames, it resulted with the QTM data being equal to the translation vector when it lost tracking. Therefore, these calculations were only done when the QTM data mismatched the translation vector. For translation, the differences between the estimate from the OpenCV and QTM positions were calculated and plotted as functions of QTM distance in each direction. All rotation plots were only done with the indices where QTM was tracking.

The orientation was first compared between the two solutions and QTM in two figures with  $3 \times 3$  grids of subplots, where each subplot corresponded to the element of the rotation matrix. This enabled a comparison of each rotation matrix element as it was shown as a function of time. Another plot was made for each solution, illustrating the difference between the estimated matrix and the QTM matrix as a function of time.

As seen in section 2.4.3.2, the pose estimation from the fiducial markers introduce a certain ambiguity, requiring more information to decide the more correct solution. The chosen option for providing information to choose accordingly, was to compare each solution with the orientation from the IMU. Some of the remaining rotation plots were made with this in mind, plotting the vector closest to the measured orientation from QTM, simulating the IMU.

The next type of plot was a representation where the rotation matrix between the closest estimated attitude from the ArUco marker to QTM and the attitude from QTM was converted to angle-axis form and the angle extracted, similar to the method for initializing the orientation of the calibration plate. This displays the error of the attitude solution closest to the QTM estimate by comparing the angles between them and the QTM estimate, frame by frame. The next set of plots were made for illustrating the azimuth angle by projecting the x and y axes down onto the XY-plane and using trigonometry for calculating the angle from the x-axis and from the y-axis, respectively:

Listing 4.5: Pseudocode for XY-Projection of azimuth angles

```
% Calculate azimuth angle from XY-projected drone X-axis
xAngle(i) = atan2d(x_y, x_x);
% Calculate azimuth angle from XY-projected drone Y-axis
yAngle(i) = atan2d(y_y, y_x);
```

As seen in figures 4.10 and 4.11, the body frame's x and y axes were projected onto the xy-plane and decomposed into each x and y component in the camera frame. As shown in listing 4.5, the angle was calculated using the four quadrant inverse-tangent "atan2" (in degrees) to get the complete angle. Since the function returns the angles on the interval  $\psi \in [-180^\circ, 180^\circ]$ , some of the recordings would

yield an inconsistent plot when the angle is close to either boundary. Therefore, logic was added to enable or disable wrapping with the modulo operator for each angle, as well as axis limits to show the tracked movement. This is shown with the highlight in listing 4.6.

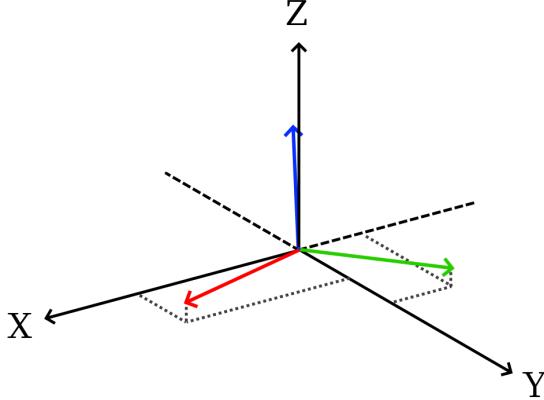


Figure 4.10: Projection of body frame onto xy-plane

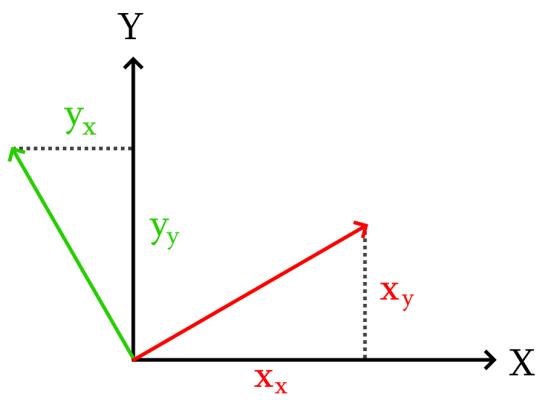


Figure 4.11: Projected x and y axes

Listing 4.6: Highlight for modulo wrapping

```
% Enable/disable modulo wrapping of angle and set limits
wrappingX = false;
wrappingY = false;
xStart = -270;
yStart = -360;
if wrappingX
    % Trig with modulo wrapping
else
    % Trig w/o wrapping
end

if wrappingY
    % Trig with modulo wrapping
else
    % Trig w/o wrapping
end
% Plot x-axis angle
ylim([xStart (xStart+360)])
% Plot y-axis angle
ylim([yStart (yStart+360)])
```

This was done in three plots, where the first two compare the two solutions to the QTM estimate separately. The third plot displays the XY-projected angles of the vector closest to the QTM solution, with a similar statement to the aforementioned axis-angle plot, together with QTM measurements. Finally, the angle between the iterations chosen vector and the QTM estimate was plotted as a function of time as well, to illustrate the potential of the machine vision system with the fiducial markers azimuth angle estimate.

#### 4.7.4 Translation Distribution Fitting

Prior to designing the Kalman filter, a script (shown in appendix G.2.7) was made for fitting distributions to the measured error between the estimated translation vector and measured values of QTM from the pose validation test. Based on the same comparison as before, only the first solution of the translation vectors were used. First, the data from all tests but #3 were used to plot the difference between the estimated position and QTM, as a function of QTM position separately for x,y and z. Like previously, the data was only used when the translation from QTM was measured

by utilizing the indices where it was not equal to the translation vector to relate the reference frames. To use this selection of data, they were appended to a new set of lists and sorted based on the QTM data to correspond with the distance. Each direction ( $x, y, z$ ) was split into  $n$  intervals and fitted to individual distributions using the distribution fitting method with 'Normal' specified. As the distributions give information of the expected value  $\mu$  and standard deviation  $\sigma$ , they were used to plot the confidence intervals with  $\pm 1 \cdot \sigma$  with grey squares. At last, the standard deviations were plotted as functions of the average of each interval with the Curve Fitter in MATLAB. This was done so the standard deviation of the positions could be parameterized as a function of the position, enabling a more controlled state estimate in the future. The R-square and RMSE values were noted from the statistics list in the application.

#### 4.7.5 Position Kalman Filter: 1 DOF

To demonstrate the possible performance of the vision system, a one dimensional Kalman filter was made with the logged data of the translation in  $z$ -direction. The filters design is inspired by the approach of Farrell's book on aided navigation [37], where high rate sensors are used as the input parameter  $u$  and the low rate sensors are used as the measurement parameter  $y$ . For the designed Kalman filters in this and the subsequent section, the QTM measurements were used for making a simulated accelerometer from an IMU. As the QTM system lost tracking during the recordings, the script was set up with linear interpolation to serve data with a rate of 100 [sps].<sup>3</sup> The speed and acceleration from the QTM data was numerically calculated as shown with the highlight in listing 4.7.

Listing 4.7: Highlight from 1D Kalman filter: numerical differentiation

```
% Save Recorded Speed z'
for i = 2 : length(t)
    zDotQTM(i) = (zQTMi(i) - zQTMi(i-1))/dt; % [m/s]
end
% Save Recorded Acceleration z ''
for i = 2 : length(t)
    zDotDotQTM(i) = (zDotQTM(i) - zDotQTM(i-1))/(dt); % [m/s^2]
end
```

The simulated accelerometer signal was created by adding a noise signal to the acceleration above. The noise signal was created as a zero mean white Gaussian noise with the `wgn()` function from MATLAB, with the power specified as a linear value based on the "Output Noise Density" from the BMI088 datasheet [15]:

$$ond_z = 190 \left[ \frac{\mu g}{\sqrt{Hz}} \right] \rightarrow w_z = 190 \cdot 10^{-6} \cdot \frac{\sqrt{f}}{g} = 1.9375 \cdot 10^{-4} \quad (4.3)$$

where  $g$  is earth's gravity and  $f$  is the frequency of the signal. To isolate issues while developing the filter, the speed and accelerations were plotted in separate figures, as problems became more quickly apparent there compared to observing the filter output directly. Initially the filter was used with a constant scalar value for the measurement noise  $R$  and replaced by the polynomial when the filter was working correctly. The process noise  $Q$  was calculated with the `cov()` method (but could also have been done with the `var()` method or the squared result from the `std()` method). The state space model chosen for the filter was set up with kinematic equations for translational motion, from classic mechanics:

$$z_k = z_{k-1} + \Delta t \cdot \dot{z}_{k-1} + \frac{\Delta t^2}{2} \cdot \ddot{z}_k \quad (4.4a)$$

$$\dot{z}_k = \dot{z}_{k-1} + \Delta t \cdot \ddot{z}_{k-1} \quad (4.4b)$$

---

<sup>3</sup>Samples per second have the same unit as [Hz], as both are measurements of something per second. This report uses [sps] to avoid ambiguities when discussing sample rates.

The equations (4.4a) and (4.4b) were then set up for the state space model:

$$x_k = \begin{bmatrix} \hat{z} \\ \dot{\hat{z}} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_{k-1} + \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix} u_k = Ax_{k-1} + Bu_k \quad (4.5a)$$

$$y_k = [z] = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k = Cu_k + Du_k \quad (4.5b)$$

Note the dimensions of matrices  $A$ ,  $B$  and  $C$  for the matrix multiplications with  $x_{k-1}$ ,  $u_k$  and  $x_k$  respectively. The input  $u$  is just one dimension; both the previous and current estimates  $x_{k-1}$  and  $x_k$  are  $2 \times 1$ . The filter was set up to iterate through the time-list with 100 [Hz], and set the input  $y_k$  equal to the measurement if the corresponding lower rate time-list was inside a specified tolerance. From this the measurement update was only done if a measurement was inside the given tolerance. The input  $u$  and measurement noise  $R$  were updated each iteration as the simulated accelerometer values and from the squared standard deviation polynomial, respectively. A highlight of the one-dimensional Kalman filter algorithm is shown in listing 4.8.

Listing 4.8: Highlight of 1D Kalman filter algorithm

```
% Find closest value
for CViter = 1 : length(time)
    if (abs(time(CViter) - t(i)) < timeTol)
        y = zCV0(CViter);
        % detected = true;
        break
    else
        y = NaN;
        % detected = false;
    end
end

% Update Input: Simulated Acceleration
u = simIMU(i);

% Update Measurement Noise
R = (p1*y + p2)^2;

% Prediction
x = A*x + B*u;
P = A*P*A' + B*Q*B';

% Measurement Update
if ~isnan(y) % if detected
    Kk = (P*C')/(C*P*C' + R);
    x = x + Kk*(y - C*x);
    P = (I - Kk*C)*P;
end
```

Finally, some metrics were calculated and plotted to illustrate the increased performance; the absolute value of the difference between the measured values and the QTM values, as well as the absolute value of the difference between the filtered value and the QTM values. From this, the mean and standard deviation were calculated, so the one-sigma-range for both were plotted in the same figure. The full 1D Kalman filter script is shown in appendix G.2.8.

#### 4.7.6 Position Kalman Filter: 3 DOF

With the design and implementation of the one-dimensional position estimator, a three-dimensional version was simple to implement. The kinematic equations (4.4a) and (4.4b) were used to express the translational motion in the x and y directions as well, with the implementation in the state space model as follows:

$$x_k = \begin{bmatrix} \hat{x} \\ \hat{\dot{x}} \\ \hat{y} \\ \hat{\dot{y}} \\ \hat{z} \\ \hat{\dot{z}} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x_{k-1} + \begin{bmatrix} \frac{\Delta t^2}{2} & 0 & 0 \\ \Delta t & 0 & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \frac{\Delta t^2}{2} \\ 0 & 0 & \Delta t \end{bmatrix} u_k = Ax_{k-1} + Bu_k \quad (4.6a)$$

$$y_k = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} x_k = Cu_k + \mathcal{D}u_k \quad (4.6b)$$

The implementation of interpolation and the Kalman filter algorithm was done like before, in the additional dimensions. The noise for the simulated IMU values were calculated like in (4.3) with new values from the datasheet [15], as they were not the same for the x and y directions as the z direction:

$$ond_x = ond_y = 160 \left[ \frac{\mu g}{\sqrt{Hz}} \right] \rightarrow w_{\ddot{x}} = w_{\ddot{y}} = 160 \cdot 10^{-6} \cdot \frac{\sqrt{f}}{g} = 1.6315 \cdot 10^{-4} \quad (4.7)$$

The corresponding process noise matrix  $Q$  was made with their variances on the diagonal. Likewise, the polynomials for the measurements' standard deviations were calculated and squared before constructing the measurement noise matrix  $R$  with them on the diagonal, on iterations with a detected marker like before. The full script is shown in appendix G.2.9.

## 4.8 Development of a server for the drone arena

### 4.8.1 Concept

There is a vast amount of data-streaming that needs to be happen from one part of the arena system to another or between processes, an example of this is sending the data from the camera to the Kalman filter along with the IMU data from the drone, then sending the output of the Kalman filter to the RL, which then generates the values for the drone motor which must be sent to the drone, and all of must simultaneously be synchronised and working in real-time. All of the messages which must be sent around the arena can be seen in figure 4.12.

### Message transmission requirements in the arena

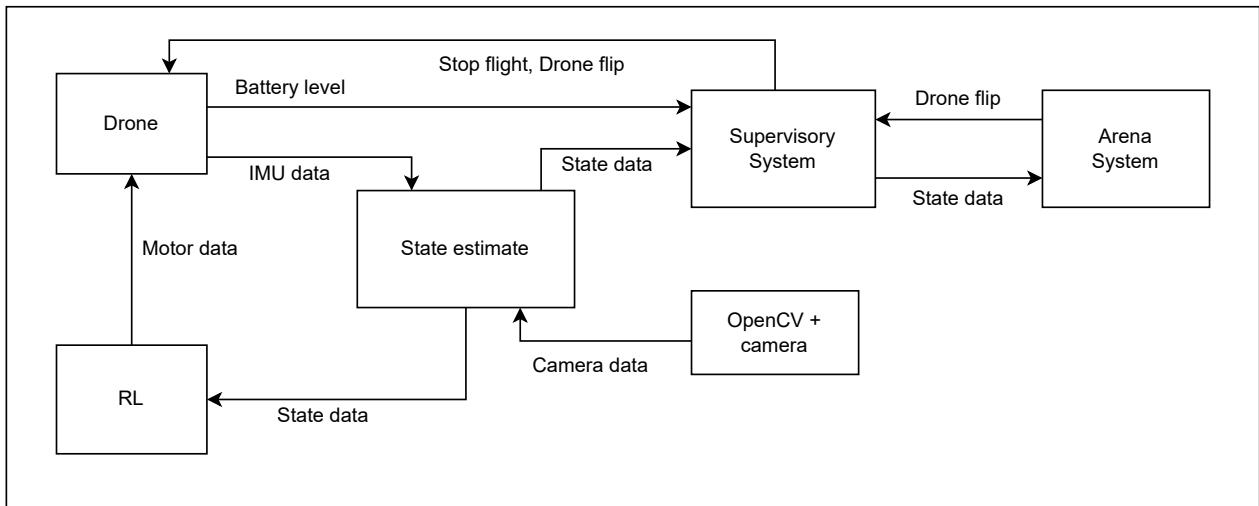


Figure 4.12: Figure showing how messages or data-streams are required by the systems within the drone arena

For this system there were a few methods that were considered, either to use ROS2 for all the communication, to develop a server system specifically for the drone arena, or to send messages directly from one process to another with no middle-man. The following explains the working method, advantages and disadvantages of these methods:

### ROS2

ROS2 could be used as the back-end for this system as the messaging method using ROS2's message interfaces along with the ROS2 tools such as topics for data-streams or services for single messages, with this method each process would have a node associated to it. An advantage of using ROS2 is that it would require writing very little code specifically for the communication so most of the programming time can be spent on the program itself, and ROS2's communication is already set up for both network communication and IPC. However ROS2 is a very large API and might offer worse performance than a more custom server code.

### Server

Creating a server which can communicate with all processes and then transfer data from one to another was a concept that was considered, this way the server can act as both the functional part of the HMI and also allowing for near perfect synchronization between the processes, this also gives the ability to change many options of the system either at the start or on runtime without any change in the code. A disadvantage of this system is that there would be a small delay for a message to arrive as a message first has to be sent to the server, and then passed on to the process which requires it.

## Direct Messages

With this concept, messages are sent using sockets from one process directly to the other processes which requires the information. This would offer the best performance of the methods discussed, however it would be difficult to synchronize the processes together or to design a HMI which can communicate to all of the systems in the arena for the user to change options during runtime. Synchronization and a HMI could be done similarly to the server method while still having data-streams being sent directly, however some data-streams need to be sent to multiple other processes, or the program to send messages to might alter on runtime as the user changes an option in the HMI, which would make each of the processes to be very code intensive for it to work.

Initially, the final direct messages concept was discarded as it would be too difficult to synchronize the processes together, and for it to be fully functional with a HMI then a program similar to the server concept would have to be created anyways. Afterwards, ROS2 was explored and a map of all the required nodes that must be created was made, this can be seen in Figure 4.13.

## Concept - required ROS2 nodes

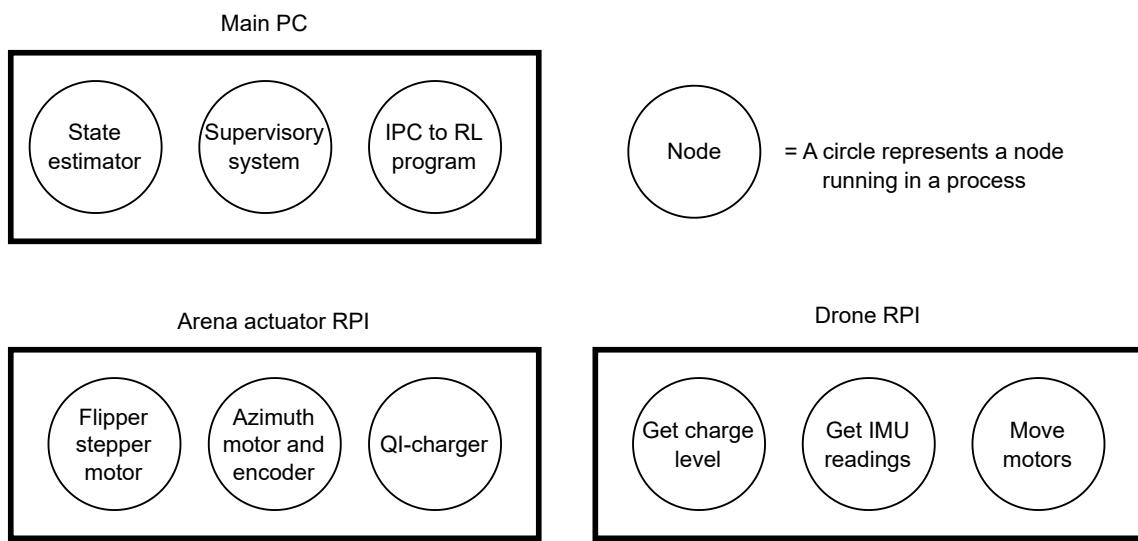


Figure 4.13: Figure showing all the required nodes for the ROS2 communication concept

The nodes in the figure in 4.13 would all have to have their own unique methods of sending messages such as topics, services, and actions depending on the purpose. Therefore ROS2 was not chosen as creating 9 separate nodes each with their own coding would be incredibly inefficient. However, ROS2 was still chosen to be used for the actuator part of the arena as ROS2's messaging system would work perfectly for separating each of the actuator components such as the stepper and servo motors into their own processes with their own nodes. A large disadvantage of using ROS2 is that it would not be very compatible with the programs running the RL, and may have a sharp learning curve for students looking to use the arena as a learning platform for RL, so possibly the best method would be to have a node which uses IPC to send converted ROS2 messages to an RL program. Therefore, it was decided to create a more general server system for the drone arena, where each of the processes would not require their own coding for communication, requiring much less time spent coding as well as better expandability in the future.

### 4.8.2 Working method of the server system

#### 4.8.2.1 Server

A concept for the working method of the server was created which is shown in Figure 4.14.

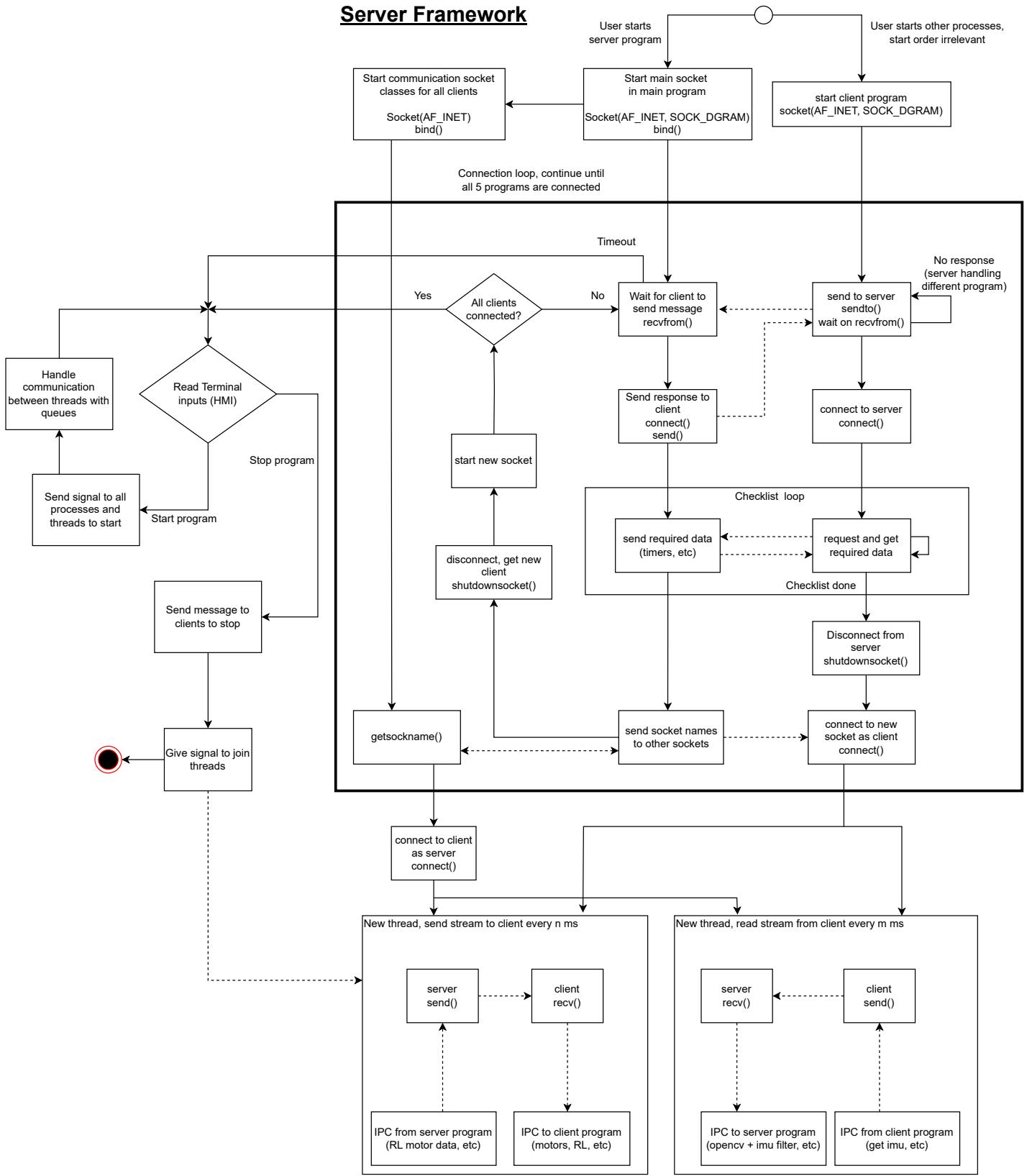


Figure 4.14: Flowchart showing the working method of the server framework

When the server program starts, it begins a main server socket which binds on a known port along with multiple other sockets which will be called secondary sockets. These secondary sockets are more specific for the processes of the arena, such as one secondary socket for the camera process

which receives a data-stream of the drone position from the camera. Since the server is entirely using UDP, the server starts with waiting on a `recvfrom()` function, after a datagram is received with valid parameters, a connection is established which begins the client's checklist loop. During the checklist loop, the client checks if it is missing any information needed to begin the program, and will send a request to the server for the information it is lacking. This includes the synchronization information, and info on the next secondary socket it will connect to for sending data-streams. Once the checklist loop is completed, the client and main server socket shut down their connection and the client proceeds to connect to the secondary socket associated with it. The main server then restarts a socket on the same port and waits on a receive from another process again.

During the connection loop, if the main socket's receive times out during the main connection loop or if all the clients connected, one or more threads are started for each of the secondary sockets with receiving or sending methods. Afterwards, a terminal HMI interface is started which allows the user to abort or start the program, at this time, the threads for the secondary sockets wait for a signal from the main thread to begin. If the user starts the program, the threads are started and a start-of-transmission message is sent to each of the clients which begin their sending or receiving methods. If the user aborts the program either at the start or after starting, an end-of-transmission message is sent to all the connected clients, and the loops in the secondary socket threads are first exited and then joined.

#### 4.8.2.2 Client packages

An easy to use package was created that includes a client method which connects automatically to the server and synchronizes with it. This would allow it to be easy to incorporate communications into the RL programs that future students would create for using the arena as a learning platform, in which a function could be called that automatically sets up the communication to the server, this way options would be controlled exclusively from the server/HMI. To prevent writing extra code the package was also be usable for the other primary functions of the arena requiring communication such as the IMU and motors on the drone. The client package was created for the programming languages C++ and Python. These were chosen because C++ would be used for controlling the drone as well as other components, and python for the camera using OpenCV as well as the RL program which would likely use Pytorch.

For the working method, on startup the client first creates a socket and resolves the server's hostname to get the IP address, it then sends a message to the server and waits for a response, if waiting for a response times out, it resend a message and waits again. The timeout can happen either if the server program has not been started yet, or if the server is in the process of handling another client, this means that the start order of any of the clients or the main server is irrelevant. If a response is successful, the client connects to the server and begins its checklist loop which goes through the same steps discussed earlier. Once it has completed the checklist loop and connects to its specified secondary server socket, a thread is created which waits on a start-of-transmission message.

#### 4.8.3 Creating the server system

This section describes the actual implementation of the server system using an object-oriented approach. The code for these programs can be found in appendix G.3.

##### 4.8.3.1 Protocol Buffer Messages

Due to the fast required speed needed for data packaging, Protocol Buffers 3 was chosen as the human readability of the data was not considered as important. Three message types were created using the protocol buffers .proto code, one for motor values and a kill switch for the drone, one message which included fields for data that would be transferred from the primary server socket to the client such as synchronisation info or IP info, and a final message for the position of the drone. Enumerators were also created for the messages which included terms such as what type of data in

the checklist was requested, an identifier for the client, and the source of the drone position data. The position message used repeat fields to create arrays, with the intention of flattening a size  $m \times n$  matrix to a  $1 \times n \cdot m$  list and specifying the size of the  $m \times n$  matrix shape in another field. This method was chosen so that it would be easier to send different types of data in the same fields, such as the IMU sending a  $1 \times 3$  vector of its data without pre-processing, or the camera being able to send a  $3 \times 3$  rotation matrix. To compile the .proto code to a usable format for C++ and Python, the protoc compiler had to be installed. It was found that the Protoc compiler included in the apt package was incompatible with the newer Protocol Buffer version from pip, however after downloading a new version from the Protocol buffer Github repository it was found to still be incompatible with the Python package from pip which also used a more updated version, so it was decided to use a downgraded version of the pip package to have the same versions as with apt, this was done with the following terminal commands:

```
$ sudo apt-get install protobuf-compiler
$ protoc --version
libprotoc 3.12.4
$ pip install --force-reinstall -v "protobuf==3.12.4"
```

To compile the .proto code using the protoc compiler, the following was typed into the terminal, where -I specifies the location of the .proto file

```
$ protoc -I=. --cpp_out={path} {filename}.proto
```

As the project expanded, a bash script was created that automatically recompiled the .proto code for all the required programs. The python and c++ compilation commands were separated so that it was easier to edit in the future. For individual projects using C/C++, this could also be done in the CMake, however this would not be able to update the protobuf messages for python so this method was chosen instead. The contents of the bash script named genProto.sh, which was placed in the same directory as the .proto code named dronePosVec.proto:

```
echo "generating protobuf files"
protoc -I=. --python_out=. --python_out=../Camera ...
--python_out=../RL/Modules ./dronePosVec.proto
protoc -I=. --cpp_out=../Drone/droneIMU/src ...
--cpp_out=../cpp/arenaServer/src ./dronePosVec.proto
```

This was made executable with the following terminal command:

```
$ chmod +x ./genProto.sh
```

The protocol buffer message code dronePosVec.proto which was used can be found in Appendix G.3.1.

#### 4.8.3.2 Synchronisation method

Before describing the complete implementation of the server, the synchronisation method of sending and receiving messages will be explained. The synchronisation of all the processes is based on a point in time of the server's monotonic timer, and this point in time is obtained during the start of the server program. If a message is desired to be sent every 100ms then the exact sending time is relative to the server's timer, such that if the server's timer is at the computer's monotonic timer time of point of 31[ms], the messages will be sent at the time points of 131[ms], 231[ms], and so on. The way this was done to make the thread sleep until it is time to send a message, an equation was made for this to calculate the length of time in nanoseconds to sleep for, which is shown in Equation 4.8:

$$T_{sleep} = T_i - ((T_{Now} - T_{Server}) \bmod T_i) \quad (4.8)$$

where:

$$\begin{aligned}
T_{sleep} &= \text{Length of time to sleep [ns]} \\
T_i &= \text{Time of sending interval [ns]} \\
T_{Now} &= \text{Current time [ns]} \\
T_{Server} &= \text{Server's synchronisation timer [ns]}
\end{aligned}$$

Nanoseconds is used as it is a more accurate way of displaying time for computers, as floating numbers may have floating point errors associated with them. The function 4.8 first takes the current time and subtracts it by the server's synchronisation time, by taking the modulo of this difference by the interval length; a length of time is output which represents a wrapping of the interval length. For example, if the difference between the current time and the server's synchronisation time is 201[ms], then taking the mod of 100[ms] of this value will output the value wrapped around 100[ms], which is 1[ms]. Subtracting the interval length by this wrapping then gives an amount of time until the next relative server time based on the interval.

Using this method, a cumulative error will never occur such as it would from calculating the difference between the last time a message was sent and the current time, however, setting a thread to sleep for a length of time will cause an offset error which will be described later. Avoiding cumulative errors is critical for a system which will be on for possibly days at a time, such as the drone arena during RL.

#### 4.8.3.3 Server

The server's header file includes an abstract class, `SocketMethods`, which creates an interface for many commonly used socket methods and other functions, such as starting a socket, setting a timeout time, receiving and sending from the socket, getting the current time, etc. This abstract class is inherited by the main server socket class `ServerSocket` and also another abstract class `AbMessenger` which is then inherited by the specific classes for the secondary sockets. The class `AbMessenger` also includes many functions used for threading.

The server program first starts with resolving the current computer's internet IP using the `getaddrinfo()` function along with `gethostname()`, it also appends the local network's name search to the input of `getaddrinfo()` which in this case is ".uia.no", appending the search name gets the computer's internet IP rather than local the IP (127.0.0.1). The program then sets up queues which the secondary sockets use to transfer data between themselves in a thread safe manner. The global server timer which will be used for the synchronisation is then set and the main server class is initialized, and a vector of the initialized secondary sockets classes called `Messengers` in the program is created. The vector uses polymorphism as it uses type `unique_ptr` of the `AbMessenger` class, but the contents of the vector includes a different `unique_ptr` type, but this is possible since they inherit the abstract class with no new unique functions, however they can override existing virtual functions of `AbMessenger` so that they have unique working methods [115].

The main socket is connected by first using `getaddrinfo()` to set up an `addrinfo` struct which can be used for binding. If the port number is not 0, then the value is converted to a char string of the port number which is the requested format of the port number for the `getaddrinfo()` function. A socket is then setup using the generated `addrinfo` struct, as well as the `SOCK_DGRAM` and `PPROTO_UDP` which specifies it is using the UDP protocol with datagrams. `SOCK_DGRAM` is ORed with `SOCK_CLOEXEC` to prevent race conditions in multi-threaded environments [61].

After creating the socket and binding, the connection loop of the main socket then starts which waits on a `recvfrom()` and checks if the received message is valid, checking the message is done to prevent errors from things such as port scanning. Once a valid message is received, a message is then sent to the sender to confirm the receive and a connection is established which begins the checklist loop as mentioned.

The synchronisation process of the checklist is shown in the figure 4.15

During the synchronization process, the server sends a message synced to 100[ms] of its own monotonic timer as explained with function 4.8, and once the client receives the message it notes down the time of arrival to obtain the server timer but in its own monotonic timer, then the client sleeps until 100[ms] relative to the timer and sends a message to the server. The server calculates a predicted time of when the message is expected to arrive, and takes the difference of the expected and actual arrival times and sends a message including the difference/offset of time to the client so that it can sync up its own monotonic timer to the server better. The offset method was chosen as setting a thread to sleep does not guarantee that it will sleep for the specified amount of time, only that it will sleep for a minimum of that time [114].

The difference of the predicted time and the actual time includes the time taken for the thread to exit the sleep, and also the time it takes for a message to be sent through a network twice, although these lengths of time are not static.

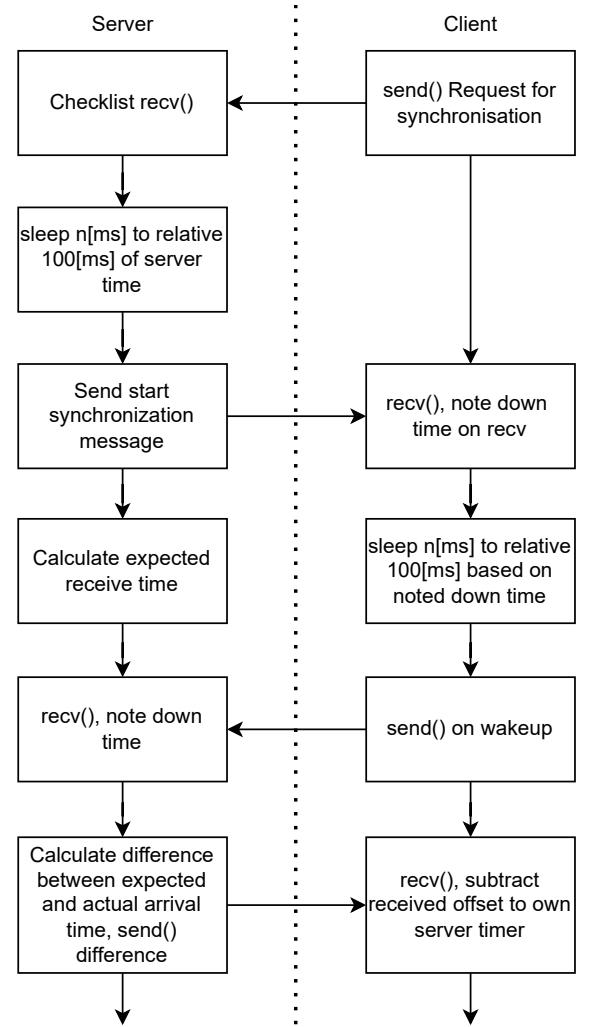


Figure 4.15: Flowchart showing the server synchronising the client's timer

To pass the IP and port of the secondary socket that the client will connect to next, a range-based for loop is used which takes a reference of a class within the vector of `AbMessenger` classes and checks if the name of the selected class matches the name of the client which the server is communicating with, if so, a function is run which populates the fields of a protocol buffer message with its own IP and port values. To convert from the binary network format of storing ports and IP addresses such as in C++ to the standard human-readable form which is used for Matlab and Python, the `inet_ntop()` function is used for the conversion of the IP, and `ntohs()` is used to convert the port number by first casting to `sockaddr_in*` which is a struct used to initialize an `addrinfo` struct manually. Along with the standard IP address and decimal port format, the binary network format along with the internet family is still sent for clients using C++.

Once the checklist is complete, both the server and client shutdown their connection to each other any the client connects to the secondary socket, and the server restarts its connection loop. After the socket is shutdown, one or more threads are started which lock using a `unique_lock` which waits on a `condition_variable`'s `wait()` function to continue, on program start, the locks are released by sending a signal to the `condition_variable` for all the secondary socket classes using a similar range based for loop as used earlier. The threads can transfer messages between each other using queues, a method was written to simulate the blocking queues function of Python which was originally intended to use mutex locks but the method was not used as it was found to be difficult to share mutex types between only two specific classes without a lot of hard-coding which was undesired, so a polling based method was used instead, although it is has worse performance. As mentioned before, the threads loop using a while loop of an `atomic_boolean` type, each secondary socket type

shares one atomic\_boolean for the threads, so that if one of the threads encounter an error and set the boolean to false, all the other threads in the class will end. The code for the server can be found in G.3.

#### 4.8.3.4 Client - C++

The C++ client package uses many of the same functions as the server. included in the header are three classes, where `SocketMethods` has the same content as the server's `SocketMethods` class, another class, `ClientClass` inherits the `SocketMethods` and includes new methods which are unique to the client package. The implementation for the methods is in another .cpp file named `clientImplementation.cpp`. As part of the `ClientClass`'s constructor, the server's hostname is resolved by using `getaddrinfo()` and then converting it to a human readable IP using `inet_ntop()`. A method is then started which connects to the IP found, which attempts a connection with the server before starting the client's checklist loop. From the client's point of view, the required information is checked to be missing, and requests are sent to the server for the data if it is.

Specifically for the drone IMU client code, an infinite loop is used for reading the IMU, however through the use of a signal handler, it can be exited with a keyboard interrupt (ctrl + c) that continues the program which safely joins the threads and ends pigpio. An example of the C++ client code being used for a drone IMU program can be found in G.4.

#### 4.8.3.5 Client - Python

The python client module was developed with a different working method than the C++ version as it was found that Python's Global Interpreter Lock was causing issues with threading working as desired for parallelism, so the multiprocessing module had to be used instead [[PyGIL](#)]. The module contains two classes, `ArenaCommunication` which includes generic sending and receiving methods for messaging, and `DataSending` which includes the main client code used to interact with the server and run the checklist loop for initialization. Python enumerators with the flag type was used to give a method of choosing the sending/receiving method by ORing the values together. The flag `GENERICRECV` sets up a sending function which is used for the camera, the flag `GENERICSEND` receives data only, and the `NO_MP` flag can be used if a custom method will be used to access the sockets, ORing `GENERICRECV` and `GENERICSEND` together will set up a method which sends and receives data. Since the multiprocessing module creates a separate copy of the program which runs simultaneously with the program, data transfer then has to happen either with signals or with queues as it is not possible to share variables between them such as with standard threading [[43](#)]. because of this, the `ArenaCommunication` class also provides a setup for using queues to transfer data, since Python uses FIFO queues, a method is provided to automatically clear the queue before writing to it so that the newest queue is always kept up to date, this is done since the rate of sending messages may be significantly slower than the program populating the queue. The module also includes a safe method of reading the queue using Python events which blocks the other process from reading the queue so that a hangup does not occur. The code `RLsendRecvEx.py` which can be found in Appendix G.5.2 which shows an example of using the client module, where it receives messages from the server and also shows an example of sending motor values to the drone.

#### 4.8.4 Using CMake and GDB

For the C++ programs, CMake was used for building projects as each executable program uses multiple .cpp files. Therefore, each package had a similar folder setup, where source code was placed in src/ and a folder was included for scripts to find packages primarily for the drone client code. Occasionally when issues occurred while developing the code, GNU debugger (GDB) was used to find where issues were occurring. To use GDB, the -g flag has to be set in the CMakeLists.txt file with *set(CMAKE\_CXX\_FLAGS "-g")*. GDB allows the code to be run line-by-line, which makes it easy to find exactly what was causing errors. When building, a new folder was created in the package directory, the following commands could be used to build the project:

```
$ mkdir build  
$ cd build  
$ cmake ..  
$ make
```

In the build folder, a .gitignore file was created including only "\*" which prevents the build folder from being pushed to the GitHub repo.

## 4.9 Arena actuator improvements

### 4.9.1 Design process

The previous group chose on-hand components for the non-3D printable parts of the actuator such as the lead screw, elastic coupling, and motors as there was little time left to order more specialized parts, therefore it was desired to calculate to see if these components were good enough or should be upgraded. This section refers to pitch as the pitch of the screw.

The vertical movement consists of a stepper motor, lead screw, elastic couple and a lead nut attached the actuator. The chosen stepper motor is a generic NEMA 17 double stack stepper motor with  $1.8^\circ$  steps, although the exact make was not found, these models are often used for 3D printers and they generally give a maximum holding torque of 0.4 [Nm] [40].

The maximum torque that the actuator will have to apply to push the actuator up can be found using the equation 2.39 by first getting the force from the weight of the actuator and the drone along with the friction from the threads and the 3 rods with bearings. The weight of the drone will be at a maximum 250g, and the actuator stick was weighed to be 0.3[kg], using  $F = m \cdot g$ :  $F = (0.25 + 0.30)[\text{kg}] \cdot 9.81[\frac{\text{m}}{\text{s}^2}]$  the force  $P$  was found to be 5.4[N]. The lead screw used was an American ACME standard lead screw with a major diameter of 0.313[in], therefore the thread pitch was 0.071[in] or 1.803[mm], with a pitch diameter  $d_P$  of 0.277[in] or 7.04 [mm] [77]. From one rotation of the physical lead screw, the lead  $L$  was measured to be 4[mm]. Generally, the friction  $\mu$  of an oil lubricated thread-nut combination is around  $0.15 \pm 0.05$ , a value of 0.2 will be used as the system has lower quality components [77]. The torque required to lift the load was then found:

$$T_U = \frac{5.4[\text{N}] \cdot 7.04 \cdot 10^{-3}[\text{m}]}{2} \cdot \frac{0.2 \cdot \pi \cdot 7.04 \cdot 10^{-3}[\text{m}] + 4.0 \cdot 10^{-3}[\text{m}] \cdot \cos(14.5^\circ)}{\pi \cdot 7.04 \cdot 10^{-3}[\text{m}] \cdot \cos(14.5^\circ) - 0.2 \cdot 4.0 \cdot 10^{-3}[\text{m}]} \quad (4.9)$$

$T_U$  was found to be 0.00765[Nm] which means that the stepper motor is adequate.

#### 4.9.1.1 Azimuth sensor

Due to the setup of the actuator system, the wires which connect to the stepper motor and QI charger would get tangled up if the actuator were to fully rotate several times. There were two feasible alternatives to resolve this, either a slip ring or programming the system in a way that it would never allow it to fully rotate more than once. For a slip ring, 6 wires in total were needed for the rotating actuators, 4 for the stepper motor and 2 for the QI charger. Without raising the height of the system, the actuator which would require a slip ring with a hollow inner diameter of 115mm so that it could be placed around the flipper stick, a slip ring with these parameters were considered to be too expensive to implement. An alternative would be to put the slip ring underneath the arena actuator, however, due to the current requirement for the QI charger of 5[A], 6 wire slip rings were too large in height for this to be viable. Therefore, the method of limiting the rotation was considered instead.

Limiting the rotation would require an absolute sensor or an end stop switch so that the actuator can know its rotation after the system powers off. The end stop switch idea was put off since with it the drone could then not be rotated exactly  $180^\circ$  if needed to.

With using an absolute sensor after the power turns on, the actuator knows its position but not how it got there, therefore it was decided that the actuator would move to a home position at  $0^\circ$  after the drone lifts off, and then rotate either clockwise  $170^\circ$  or counter-clockwise  $190^\circ$ , this way the actuator can rotate  $180^\circ$  but still know its position and how it got there, since it can only get to  $180^\circ$  by rotating counter-clockwise.

A ring encoder which could be mounted onto the rotating azimuth plate similar to the initial idea of the slip ring was considered, the ring encoders AksIM-4 Big Rings by RLS was found which fit this requirement perfectly, unfortunately these types of encoders were too expensive and would limit the budget required for developing drone [100]. Therefore it was decided to use an absolute encoder mounted underneath the entire actuator system. To not displace the plates, the plate supports

would have to be decreased in height and re-printed. A new plate was added to the bottom of actuator system to give mount an absolute encoder as well as raise the height of the rotating section so that a magnet could be placed at the bottom for reading the rotation. An AS5600 encoder in a breakout PCB was used, it was desired to use a non-metallic mounting method so that it would not potentially interfere with the reading of the magnetic sensor, so a snap-on mechanism was implemented into the bottom plate. A groove in the plate was created for the wires to pass through. The bottom plate with the absolute encoder mounted can be seen in Figure 4.16.

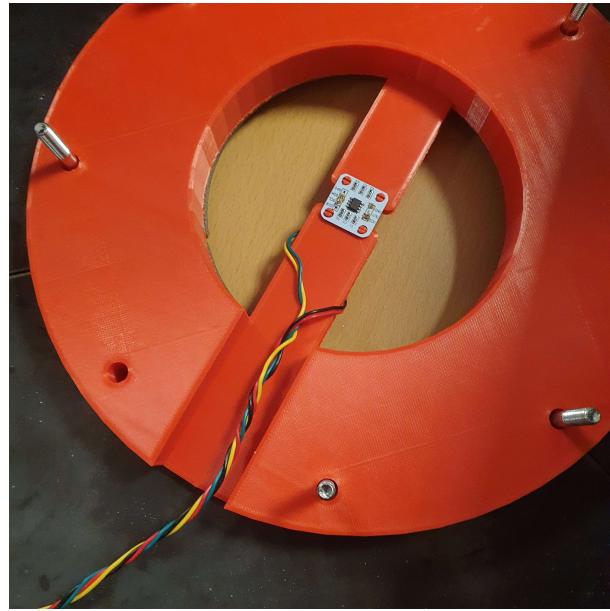


Figure 4.16: Image showing the bottom plate of the arena with the absolute encoder mounted

For mounting the magnet, it was originally intended to be glued onto the screw which is used to hold the rotating section in place. However, it was feared that the screw would interfere with the magnetic field and it was also found that this method had too high of a clearance between the encoder and the magnet. This could be seen in the reading of the status register of the encoder, which read the value 0 in the MD bit which showed that the encoder could not detect a magnet [6]. Therefore, a simple spacer was 3d printed that attached around the screw nut and pressure-fit the magnet in place closer to the encoder. The original clearance compared to the clearance with the spacer attached can be seen in Figure 4.17.

Using the standoff which increased the height of the magnet by 2mm, the encoder could then be read which showed as the status register was reading 1 in the MD bit. A simple code created with the pigpio library for reading the status register of the AS5600 as well as continuously reading the sensor, which can be found in Appendix G.6.2.

Because of the inclusion of a plate to hold the absolute encoder, the entire platform was raised 20mm, to compensate for this so that the plates still fit, the plate holder in the middle of the arena was made shorter by 20mm with no other changes made to it.

#### 4.9.1.2 Endstop switch

The previous iteration of the drone arena did not have a method of finding the location of the vertical actuator, although the steps of the stepper motor could be ran in reverse, there can still be some uncertainty to this from something such as a floating point error giving the wrong number of steps slowly over time. Therefore, it was decided to implement an end-stop switch to mark the lowest point of the vertical actuator's travel. The end-stop switch was implemented on the rotating platform although it was initially considered to be placed outside of it to prevent wires tangling, but it was found that this limited the movement of maximum travel of the platform. A holder for the end-stop button was added to the bottom rotating platform, and the vertical stick was also

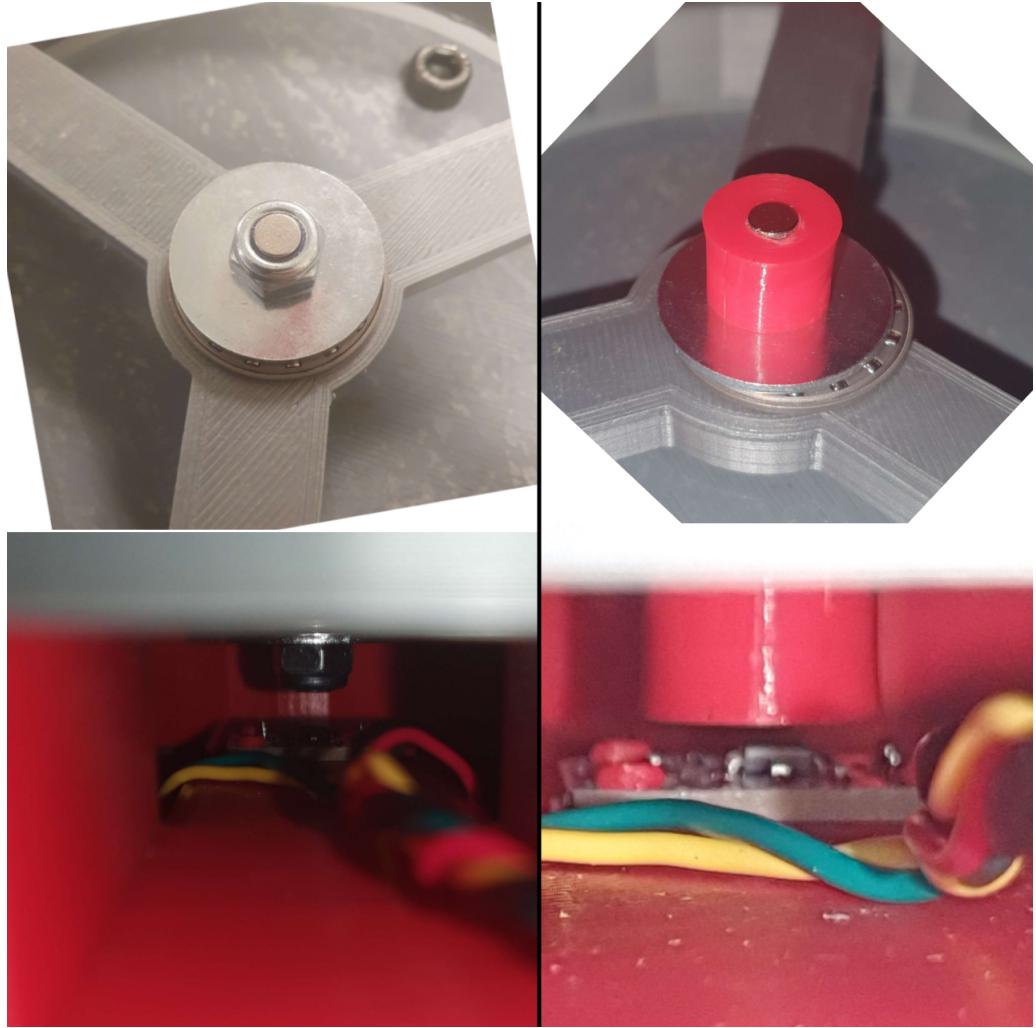


Figure 4.17: Image showing methods of mounting the magnet and the gaps between the encoder and it. The original method for mounting the magnet is shown on the left, compared to using the magnet standoff shown on the right

modified to add a cube with an adjustable height which would interact with the end-stop switch. These modifications can be seen in Figure 4.18.

For the wiring of the end-stop switch, it was desired for the button to connect to ground when depressed, which would then use an internal pull-up resistor in the GPIO pin of the raspberry pi.

#### 4.9.1.3 Communication with the actuator system and the server

Commands to flip and/or rotate the drone and charge it have to be sent from the server PC to the system which controls the actuator. Two alternatives were chosen for this actuator system, either to have a single-board computer running ROS2 as with a node to communicate using ROS2 messages, or a microcontroller which could communicate with serial USB, ethernet, or wireless. Either of these methods could perform well as the commands sent from the pc would be simple, such as rotating the drone  $x^\circ$ , flip the drone, etc, the actuator system would then handle the commands to move the actuator system.

Initial testing for the microcontroller system using serial USB was done with an ATMEGA32u4, where the LUFA "Lightweight USB Framework for AVR" library [18] was used to setup the framework to receive serial data. A test program for the ATMEGA32u4 was made using the LUFA library which turned an LED on or off based on data sent through USB, this code was modified from a demo code in the LUFA library, which was a VirtualSerial demo where the microcontroller ran as a device using the classdriver interface. The data was sent through a VScode serial communicator

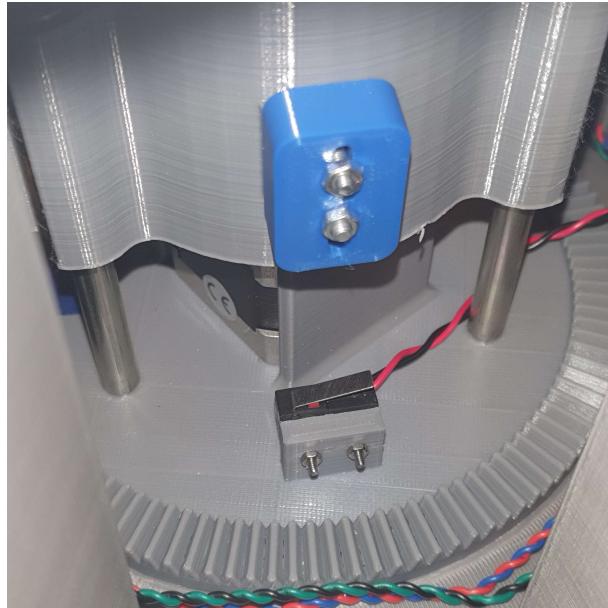


Figure 4.18: Image showing modifications on the arena actuator for the end-stop button

plugin. The results from this test can be seen in Section 5.5.1. It was then desired to test wireless communication with a microcontroller with the use of an ESP32 or STM32 with wireless capabilities, but testing with the raspberry PI single-board computer began first, which eventually took precedence due to the faster development time and capabilities of running larger systems.

#### 4.9.2 Actuator PCB

The PCB design was a continuation of the previous 2022 drone arena project. The previous PCB included the drivers for the DC and stepper motor and a port to connect to a raspberry PI. The PCB however lacked multiple components for the actuator to fully function such as an end-stop switch for the stepper motor, an absolute encoder for limiting the rotation, and a transistor to enable/disable the QI charger. A redesign of the previous PCB was chosen as an alternative to implementing a second PCB alongside the original one as this would allow for better understanding of the system.

The circuit schematic was first remade including the new components which added 4 more GPIO pins for the components, decoupling capacitors was also added to both the motor drivers based on the manufacturers recommendations. The schematic can be found in B.1. An initial version using only through-hole components was created, however a complete redesign of the PCB was done with only SMD components. Since it was known that the PCB would be soldered by hand without a stencil, mostly large-medium sized components were chosen such as size 1210 resistors and capacitors where smaller sizes could be used. In most of the other cases, the size of the components were limited by a high current and wattage requirement.

To control the QI charger, a BSC007N04LS6 mosfet was implemented as the QI charger only needs the DC input and the internal circuitry of the QI charger handles the rest of the processing. Since the mosfet will not be rapidly switching on and off, the gate switching speed is not so important, therefore the gate resistor was dimensioned after the desired current. A gate resistor of  $330[\Omega]$  was chosen to limit the gate current to  $10[\text{mA}]$ . Using the formula 2.41 and with a total gate capacitance of  $94[\text{nC}]$  from the datasheet of the mosfet, a gate resistance of  $330[\Omega]$  was calculated to give a switching speed of  $9[\mu\text{s}]$  which is sufficient for the usage [54].

Due to the addition of the QI charger which pulls  $5[\text{A}]$  continuous current [14], the previous fuse which was a RXEF300 that has a trigger current of  $6[\text{A}]$  and a holding current of  $3[\text{A}]$  had to be changed [68]. To dimension the new fuse, the maximum current that the system can pull was

calculated by adding up the continuous current pull of the components [126] [92]:

$$\begin{aligned}\text{TB6612FNG (single motor)} &= 1.2 \text{ [A]} \\ \text{DRV8825 (both coils)} &= 2.2 + 2.2 \text{ [A]} \\ \text{QI Charger} &= 5 \text{ [A]} \\ \text{Total} &= 10.6 \text{ [A]}\end{aligned}$$

This value will realistically never be reached as not all of these components will run at the same time, however to speed up the time between each drone flight, the QI charger and the azimuth DC motor using the TB6612FNG driver might want be on at same time, giving a minimum continuous current of 6.2[A]. Therefore the fuse was dimensioned after a holding current of 6.5-8[A], and a trigger current of 11-12[A]. The fuse 2920L600/12MR produced by Littelfuse was found which has a holding current of 6[A] and a trigger current of 12[A]. Unfortunately there were very few options of SMD fuses with fitting values and the fuse found was the best fitting of the available ones. Although the holding current was lower than desired, from read the datasheet it was found that the time to activate the fuse at 6.2[A] was long enough that it did not matter [67].

The previous reverse polarity protection diode which was rated at 10[A] had to be updated as well, and an SMD diode with a rating of 12[A] was found, to save cost, the same diode type was also used as the flyback diode for the QI charger which only needed to withstand max 5-6[A] [67].

For powering the components, a choice was made to use a power plane for the 12[V] traces, and due to the high current it was chosen to use 2[oz] copper plate weight rather than the standard 1[oz], which have a trace height of 0.070[mm] and 0.035[mm] respectively [58]. Despite using a power plane, the minimum trace widths were also calculated from the maximum current, for this the inbuilt calculator in KiCAD was utilized which uses the same formula for trace widths as in 2.42. The VIA sizing was also checked through the same calculator in KICAD but the standard VIA size of 0.8[mm] diameter and 0.4[mm] VIA hole was found to be sufficient and was not changed. Due to using a power plane, Thermal reliefs were added to all the components which were connected to the power plane so that it would be easy to solder, and it was made sure that the sum of the thermal relief widths added up to the minimum trace width calculated from the current that the component would pull.

Because the raspberry pi does not having an internal AD converter, it was decided to use I2C to communicate with the new absolute encoder. Due to the long wire from the absolute encoder to the arena, the I2C pull-up resistors were therefore calculated rather than using a standard value, using a rough rule of thumb that each I2C slave adds 15[pF] and every cm of wire adds 3[pF] of bus capacitance, the total bus capacitance was calculated to be 480[pF] assuming a total of 150[cm] of wiring and two components [128]. Using standard mode with 100[kHz], this gives a maximum resistance of 2.5[kΩ], and the minimum resistance was calculated to be 966[Ω], therefore a resistor of 2.2[kΩ] was chosen, which is a commonly used value for I2C [3].

The electrical components were placed primarily based on the connectors and the connection to the 12[V] power plane. The connectors were placed near the edge of the circuit board and the components which interacted with the corresponding connectors were placed nearby, the large size of the decoupling capacitors also limited where components could be placed as they need to be as close as possible to the power input. Four grounded M.2 mounts were added to the corners of the board for mounting to the rack in the arena.

The PCB acts only as an interface for controlling the motors and reading sensors, so contacts had to be added to connect the PCB to these components, as well as mounts for the motor driver boards. It was desired that the contacts would be separately sized for the different components so that faulty connections could not be caused by placing a wrong contact somewhere. Luckily, most of the contacts had separate pin amounts or current requirements, so many of the contacts reused the same model but in different sizing. For the components that would only see logic values such as the I2C and limit switch, the small ZR SMD contacts by JST were used, for the contacts connecting to the motors and the QI charger, TE-Connectivity's MTA-100 contacts were used. To connect the raspberry pi to the board, a molex milli-grid 2x9 connector was used, And finally, SMD versions of a header pin, terminal block, and sockets for the motor driver boards were also found. The bill of materials for the PCB can be found in Appendix B.1. The circuit board PCB can be seen in figure 4.19.

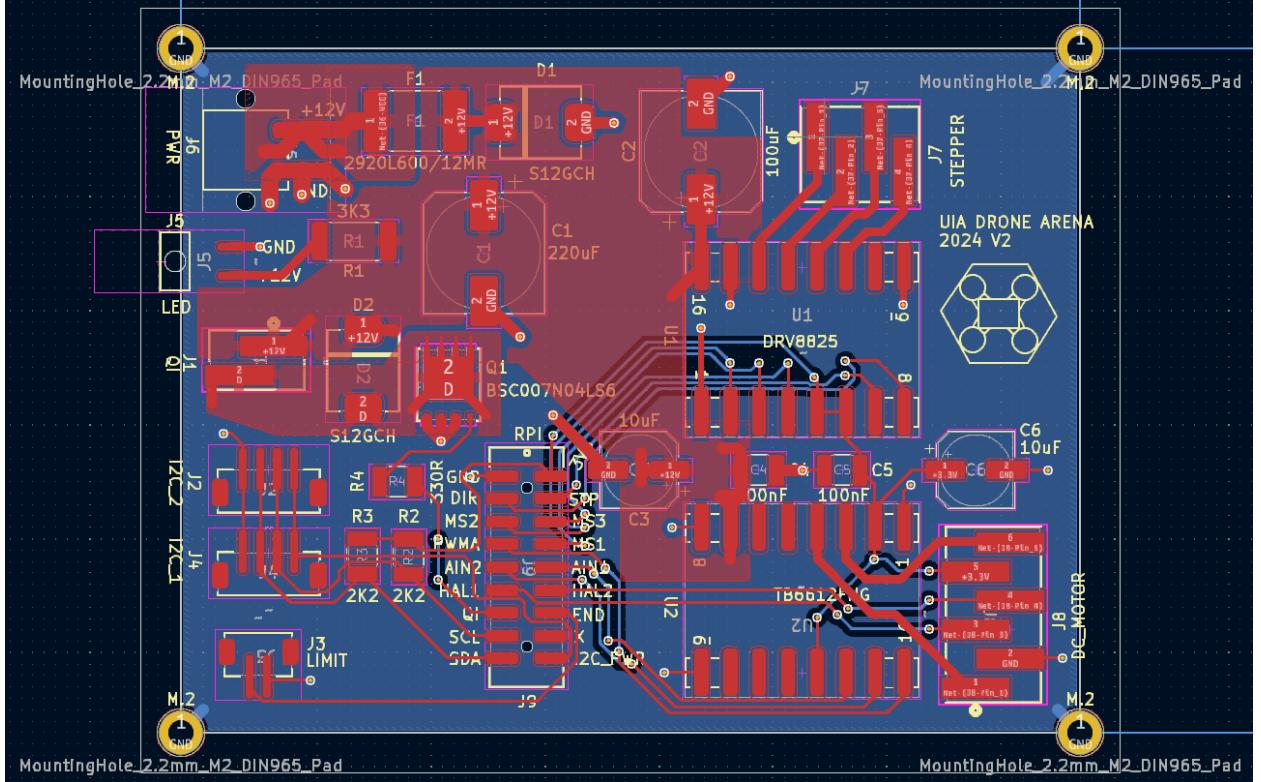


Figure 4.19: Second iteration of the arena circuit board

#### 4.9.2.1 Soldering the PCB

The PCB was ordered from JLCPCB, and it was soldered using a soldering paste dispenser operated with a foot pedal and a manual pick and place machine, then soldered with a soldering oven. Choosing large SMD components ended up saving much more time than expected during this process. During the heating process, the MOSFET slid off the pad and had to be re-soldered using a hot air reflow station. An image of the soldered PCB can be seen in Figure 4.20

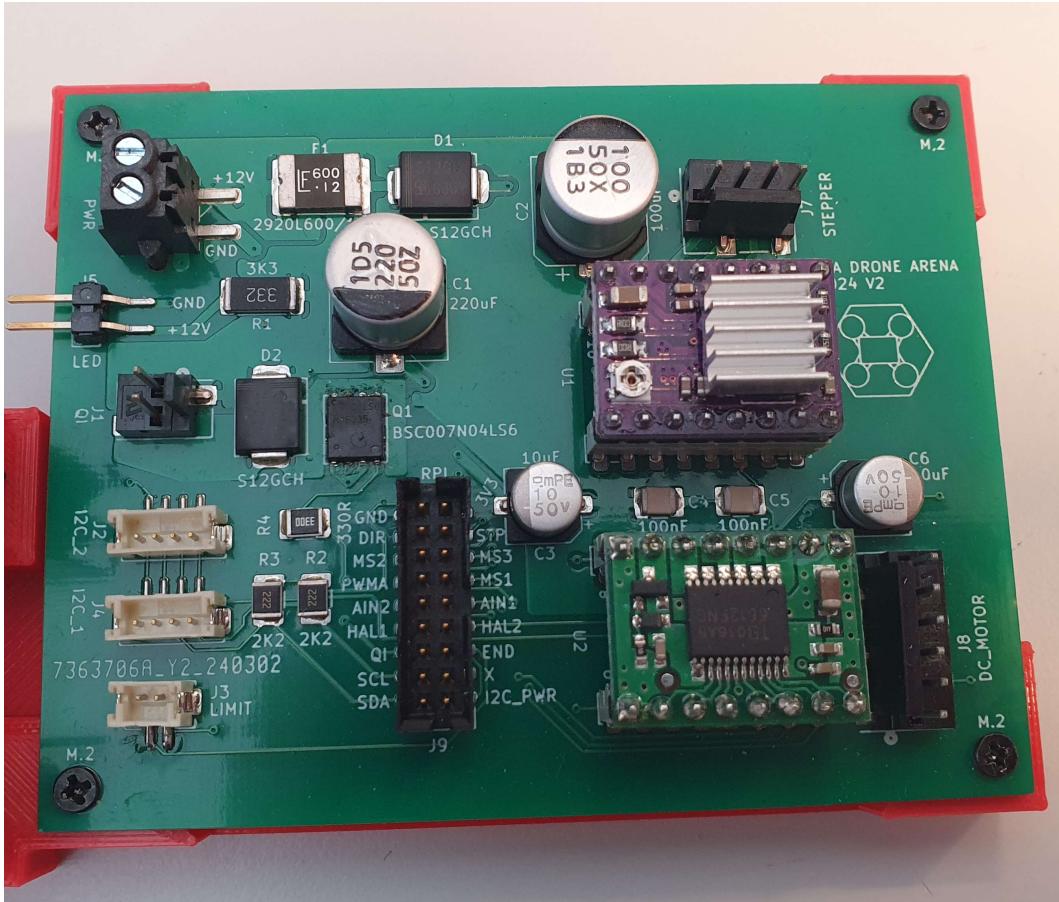


Figure 4.20: Image of the soldered PCB with motor drivers attached

The heads for the connectors had to be crimped by hand as the group did not have a high enough budget for the specialized crimping tools. The motor drivers were then placed into the sockets and heat sinks were added to them.

The pinout for the RPI INPUT connector J9 to the raspberry PI 4B+ can be found in the table 4.4.

RPI PIN	RPI CONN		RPI PIN
6	GND	3V3	1
GPIO 22	DIR	STP	GPIO 23
GPIO 7	MS2	MS3	GPIO 1
GPIO 12	PWMA	MS1	GPIO 8
GPIO 17	AIN2	AIN1	GPIO 27
GPIO 25	HAL1	HAL2	GPIO 16
GPIO 5	QI	END	GPIO 6
5	SCL	X	
3	SDA	I2C_PWR	2

Table 4.4: Table showing the connection of the J9 RPI input connector to pins on the raspberry PI 4B+

## 4.10 Drone Hardware and Software

For controlling the GPIO pins of the raspberry PI CM4, the pigpio library was chosen after also testing WiringPI for SPI, the WiringPI code can be found in Appendix G.6.4, however it was found that the documentation for the pigpio library was significantly better while functionally working the same, so the pigpio library was chosen. For running code on the CM4, the code was cross compiled from a computer running a different architecture. Pigpio was also crosscompiled to function on the the CM4, and both of these methods can be found in the Appendix F.2.

### 4.10.1 Buildroot

To reduce the demands from drone hardware in the future, a custom operating system was created. A buildroot project is established, and necessary packages for communicating with the BMI088 and connecting to WiFi were added. The steps that was done to create the OS using buildroot can be found in Appendix E.5.5.

### 4.10.2 IMU

Since the BMI088 was not ready yet on the drone PCB, a BMI088 shuttleboard was ordered. Because the shuttleboard used 1mm pitch pins rather than the standard 2.54mm pitch pins, wires were carefully soldered onto the pins while making sure that they were not touching eachother. A test was done with I2C so that the connection could be confirmed, and the i2cdetect command was used to see the connection. Once it was confirmed that the pins were not shorting and the correct I2C address was returned, SPI was then connected by following the datasheet and tested began with using the pigpio library.

#### 4.10.2.1 Troubleshooting the IMU SPI using a logic analyzer

The datasheet for the BMI088 IMU was unspecific with exactly how data was received from the MOSI line to the sensor SDI, such as if data received from SPI was read LSB or MSB first on the IMU. Reading the ID register of the accelerometer register returned the correct value, to do this the first sent bit of the byte had to be 1 which tells the IMU that a reading of a register is desired, followed by the register address which was 0x0 for the ID, this was sent LSB first, and any tests with sending MSB first and flipping the bit order of the address byte received an unexpected value. Receiving a correct value was a good start and showed that reading a register had to happen MSB first, however the order of the address for the ID did not matter, so it could not be used for any further debugging. An FX2LP logic analyzer along with the Sigrock Pulseview software was used to help debug the SPI line further as it shows exactly how data is sent through the physical SPI lines. A test was made to write a byte to a R/W (Read/Write) register, and then to read the same register again, if the address for reading or writing is wrong then the output will either be a random value or 0x0, with this test the order of the byte for writing to a register correctly could also be found. The chosen R/W register was the accelerometer's ACC\_PWR\_CTRL register with the address 0x7D, part of the initialization process for starting the IMU was to write 0x4 so this was chosen to see if the initialization could be done correctly.

To start, the sending byte was setup without any special processing, setting the R/W bit #0 to 0, and 1-7 as the address 0x7D by bit shifting it left once. The data to be written was set as decimal 4, therefore the two bytes to be sent were 0xFA04, These bytes were then chosen to be sent LSB first. In Sigrock Pulseview, this sequence can be seen in figure 4.21.

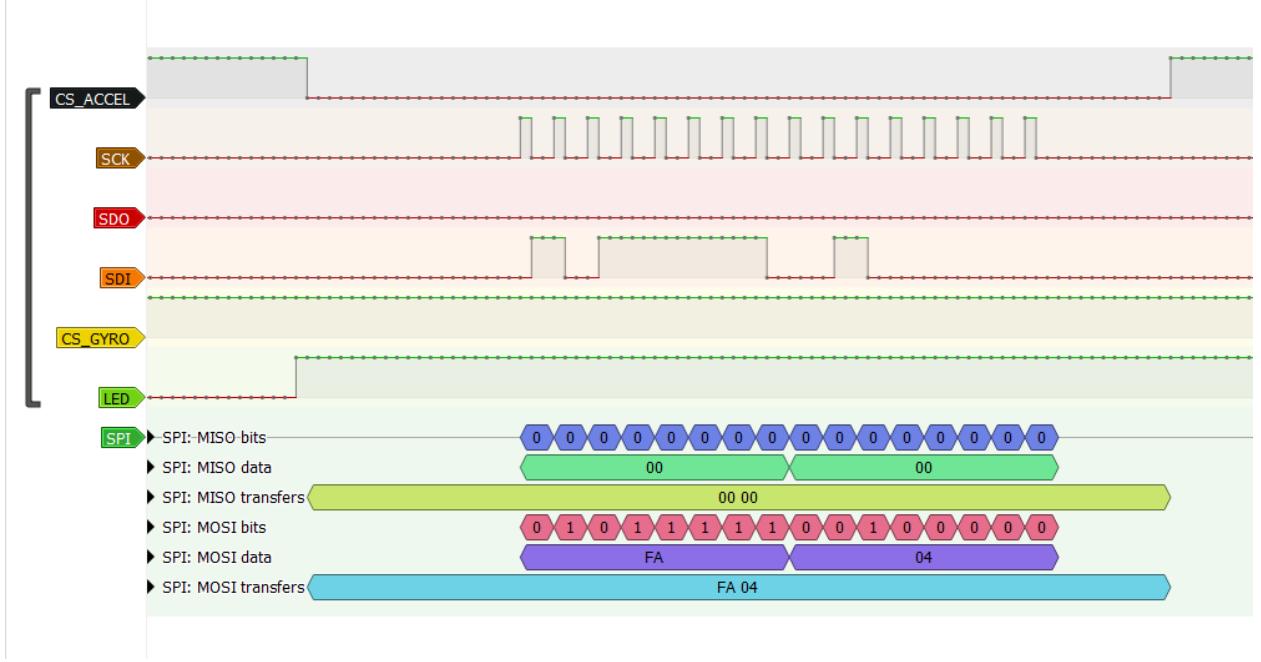


Figure 4.21: Logic analyser of sending 0xFA04 LSB first, values are shown LSB first

When reading the value, the datasheet specified to set the R/W bit to 1, followed by the same address, however, what to set as the remaining 8-15 bits of the data byte and when to read the value was unclear, therefore the second byte of the data byte was set to 0xFF, and the data was read in the next two bytes. This resulted in the bytes being sent as 0xFBFF, this showed the following in sigrock Pulseview, as shown in Figure 4.22.

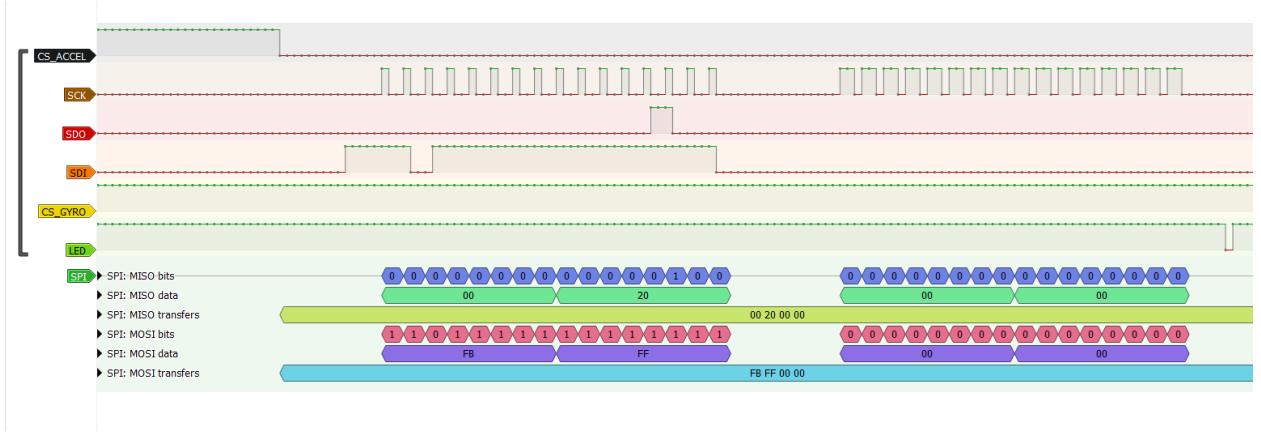


Figure 4.22: Logic analyser of reading after sending 0xFA04 LSB first, values are shown LSB first

As shown in the logic analyser, the second byte sent from the IMU during transmission of the bytes displayed 0x4 when read MSB first which was the same value that was written. Continuing to read the next bytes resulted in the reading 0x0, therefore it was uncertain if 0x4 being sent was a fluke or not. To check if this was correct, writing the value 0x0 to the register was attempted, which resulted in the exact same output even after resetting the device, proving that this was likely a fluke.

For the second attempt, the bit order of the address was reversed while still being sent LSB first, this changed the address byte from 0xFA to 0xBE, the value to be written was kept as decimal 4, this can be seen in figure 4.23.

This resulted in the same values being output. After this it was attempted to repeat the first two tests but with the writing byte be flipped, turning 0x4 to 0x20, initially, the byte 0xBE20 was sent LSB first, and the reading of the same register as shown in figure 4.24.

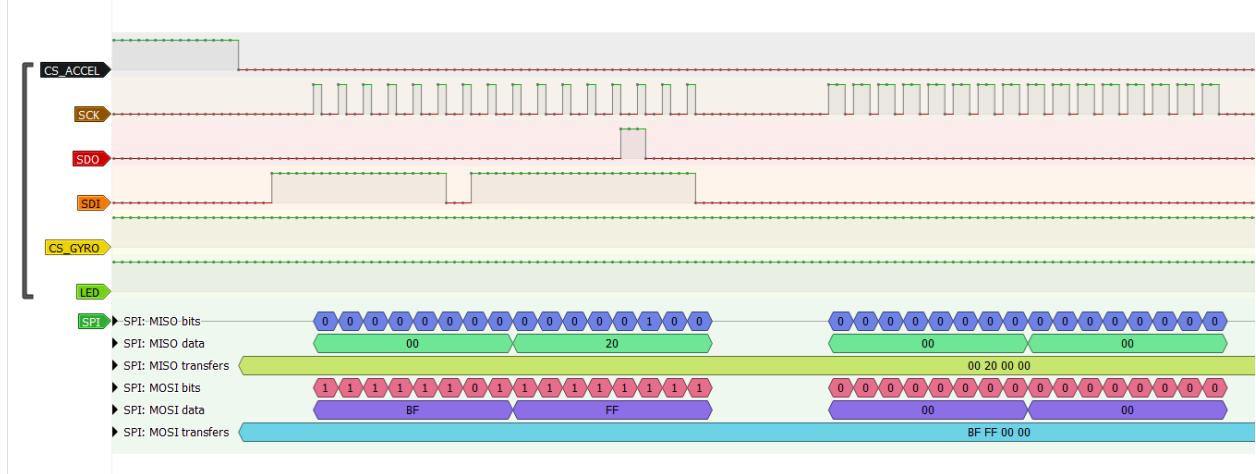


Figure 4.23: Reading with the previous input but with the address bit order flipped

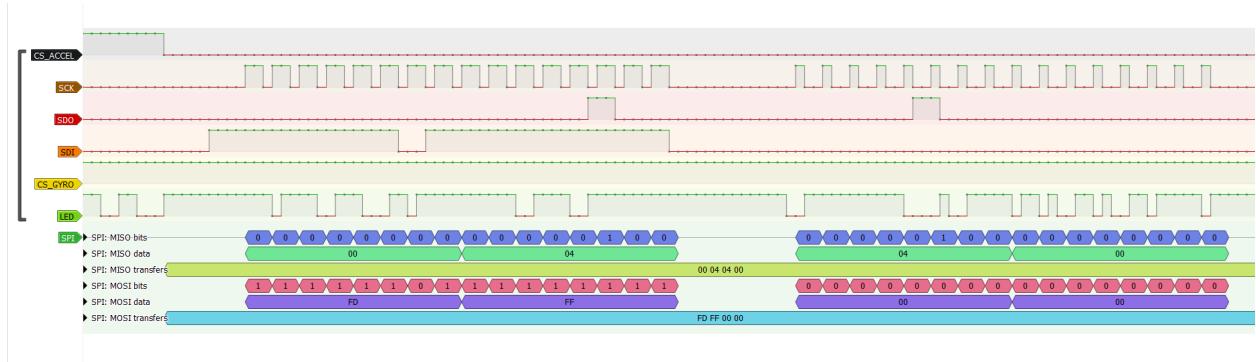


Figure 4.24: Reading after sending 0xBE20, values are shown MSB first

Terminal outputs:

```
spiHandle:0
bytes sent: 10111110 00100000
ACC_PWR_CTRL: 00000100
Gyro ID: 00010110
Accel ID: 00011110
```

This finally output the expected value of 0x4, subsequently, 0xBE20 is 0x7D04 when read MSB first, this shows that the sensor actually expected a value to be sent MSB first as is standard for SPI, therefore the initial attempts which resulted in failure from sending MSB first were likely due to a software issue.

The transmission mode was then switched to send MSB first and the tests were reattempted after resetting the registers in the sensor by disconnecting the power. For this test, the two bytes 0x7D04 were sent MSB first, and 0xFDFF (0xFD = 0x7D OR'ed with decimal 1 bit-shifted 7 times to the left) were read MSB first as done previously. The figure 4.25 shows the value of 0x7D04 being sent, and the figure 4.26 shows the values of the same register being read afterwards.

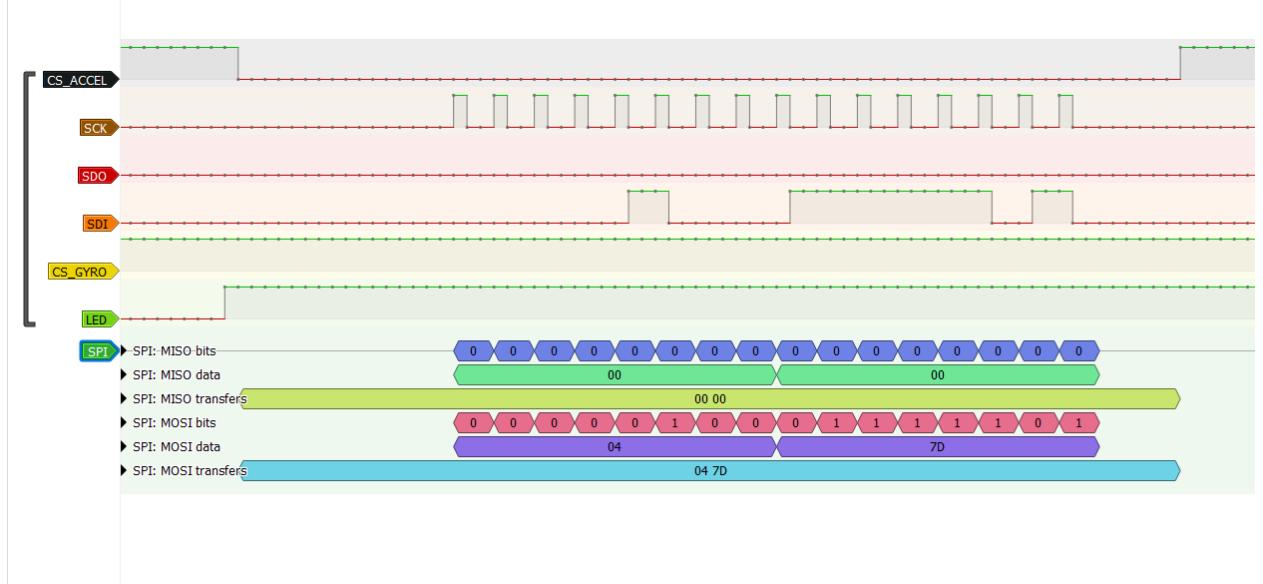


Figure 4.25: Sending 0x7D04

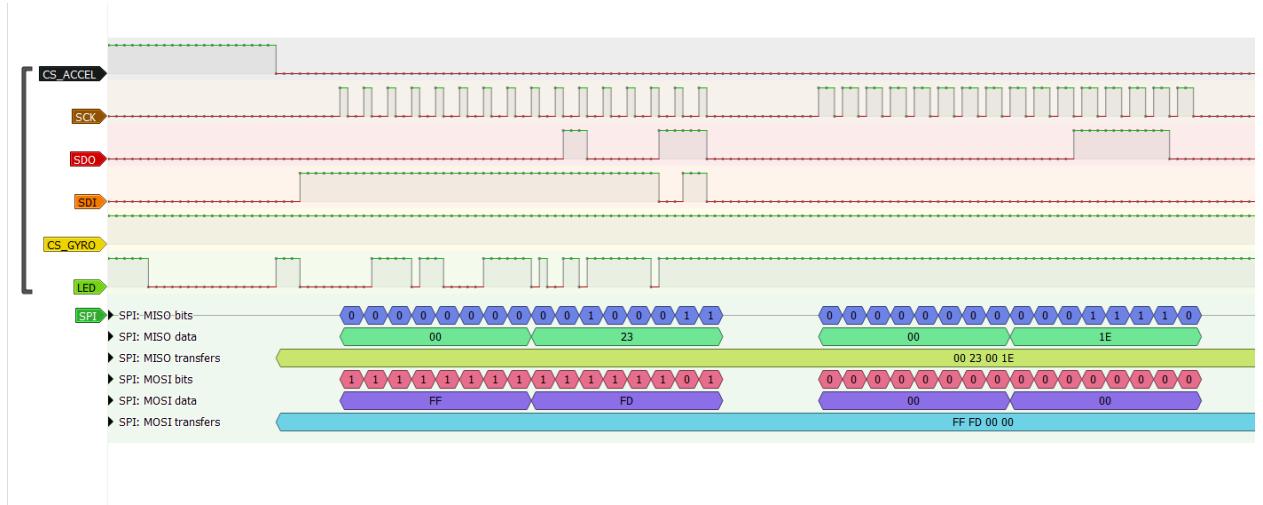


Figure 4.26: Reading 0xFDFF, values are shown MSB first

From the logic analyzer, it was shown that when tasked to send MSB first, the piggpio library actually sent the entire byte array in the reversed order, therefore sending the data byte bits 8-15 as 0-7 and the address and R/W bits 0-7 as 8-15, this explains why earlier tests with MSB resulted in the wrong values being sent. The byte order was then flipped and the same test reattempted, as seen in figure 4.27 of sending the byte, and figure 4.28 of reading the register.



Figure 4.27: Writing same as test #4 but with the byte order flipped

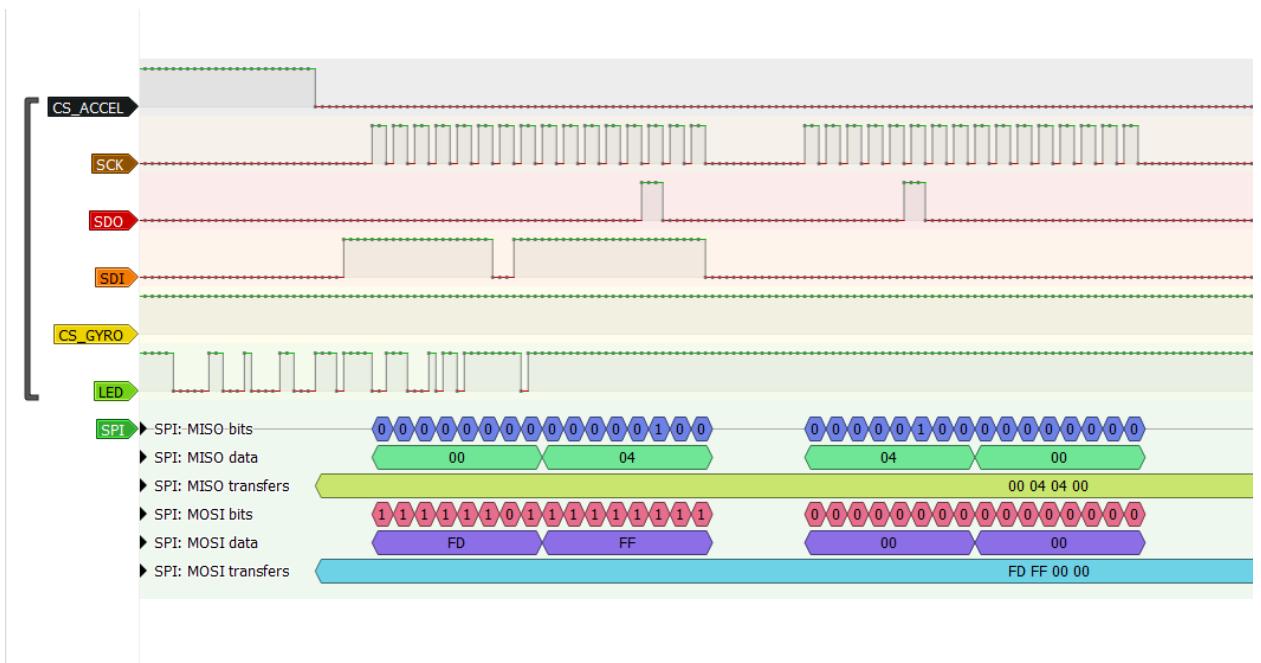


Figure 4.28: Reading same as test #4 but with byte order flipped

Which shows the correct output from the accelerometer's registers. However, when reading the gyrometer's ID, the ID read was not the expected value of 0x0F as specified in the datasheet, but the values sent during the transmission of the 0xFF byte is the value of the gyrometer ID, as mentioned in the datasheet, the accelerometer first sends a dummy byte before sending the correct register data, but the gyrometer does not. In the logic analyzer, this can be seen in the reading of the accelerometer's ID, the correct address is actually sent on the first read byte, therefore the first byte sent is likely started directly after the address byte has been sent during the transmission of the data byte with 0xFF. To read the byte during the transmission, pigpio's `spiXfer()` function

can be used instead of a combination of `spiWrite()` and `spiRead()` as done previously with the accelerometer, as shown in the figure 4.29

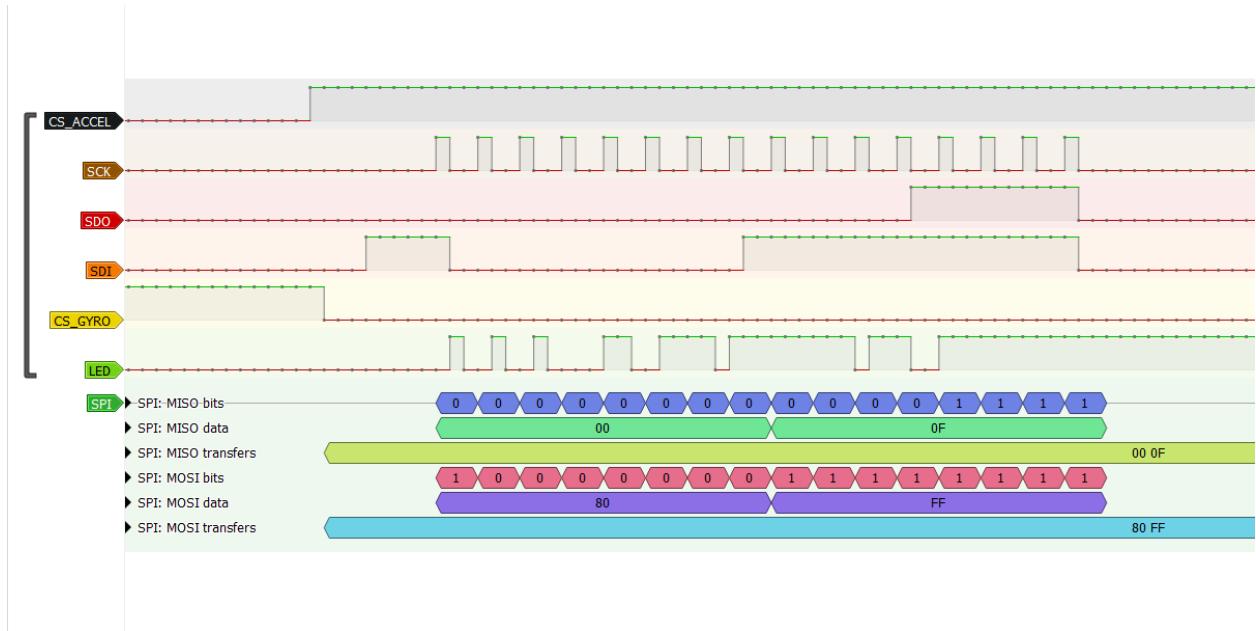


Figure 4.29: Reading the gyroscope's ID register with pigpio `spiXfer()`

Terminal outputs:

```
spiHandle:0
bytes sent: 00000100 01111101
bytes read: 11111111 11111101
ACC_PWR_CTRL: 00000100
Gyro ID: 00001111 00000000
Accel ID: 00100011 00011110
```

This returned the expected ID value of 0x0F from the gyroscope's register. As can be noted from the output commands and comparing the logic analyzer, the byte array which is output to the c++ program is in the opposite order than what was sent through the SPI lines. The final code can be found in G.6.1, with the IMU connected to SPI1 interface on the raspberry pi4 B. It is also likely that the spi interface was intended to work with 8 bit signals.

#### 4.10.2.2 Reading and calibrating the IMU

After values could be read from the IMU, a program was made for continuous reading from the IMU values, which would later be implemented with the C++ client module for sending the IMU data to the server, results from reading the IMU can be found in 5.6.1. A header file along with an implementation file was created for the IMU reading for modularity, which incorporated an OOP approach as the methods for reading the accelerometer and gyroscope were slightly different, and two classes for the two components were created. The classes constructors includes methods for initializing the sensor then calibrating them. To initialize the gyroscope, the `GYRO_RANGE` register was set to use the the scale of  $\pm 1000 \frac{deg}{s}$ , and the inbuilt low-pass filter was set to a bandwidth of 116[Hz]. The accelerometer was then initialized by setting the `ACC_PWR_CTRL` register to 0x04, which enters to normal mode of the accelerometer, and the scale of the accelerometer was set to the maximum of  $\pm 12[g]$ , and the low-pass filter was then set to an output data rate (ODR) of 100[Hz] with the normal filter setting. These initialization procedures were found from the datasheet [15]. To calibrate the offset of the sensors, 1000 values of the sensors were read and the average of the readings were subtracted from the subsequent readings. Readings were taken by reading the `ACC_X_LSB` and then continuing to read, which reads the next incremented registers. The output LSB and MSB bytes were combined to one 16-bit value by ORing the LSB value with the MSB

value that was bit shifted 8 time to the left. Using the equations specified in the datasheet, the 16-bit integer was then converted to a human readable value as well as subtracting the calibration offset from it.

## 4.11 Mechanical Drone Design

### 4.11.1 Material Choice and Polypropylene 3D-printing

As the 2022 mechatronics BSc group reflected around the material choice for the drone's airframe [14], the same material was used for this generation of the drone. As polypropylene (PP) is very lightweight and ductile, it can absorb the impacts of the flight cycles without compromising weight. However, a more stiff drone frame is desired to keep the motors in place, where PLA (polylactic acid) was used as it is lightweight, simple to print and experiment with during prototyping.

Since polypropylene (PP) is highly susceptible to warping during printing, an enclosure was crafted to ensure a stable ambient print temperature. This was sewn together from primarily Beaver Nylon and a small transparent oilcloth for visual inspection during printing, outside a support made of plastic tubes. The Beaver Nylon is a windproof material for outdoor use, suited for the application as it keeps the heat inside the tent. Instead of experimenting with printing the entire drone hull, a small benchmark print was created to evaluate different 3D printer settings. The Artillery X4 Plus printer was used, which does not support the PP filament natively, so experimentation to find an appropriate bed and nozzle temperature was required. Analyzing the print bed temperature with an infrared temperature sensor revealed that the bed temperature was not completely homogeneous across the entire surface, which was taken into account when slicing and positioning the print model on the print bed. Due to the warping issue, polypropylene benefits greatly from an adhesive surface. In order to secure attachment to the print tray, a layer of Tesa packing tape (Tesa 4280) was applied to the print tray. A generous raft setting was also used to keep warping from distorting the printed part. The top part of the hull was difficult to separate from the raft, though supports were easier to remove. Therefore, a small 1 [mm] tall nipple was added to the surface of the model facing the print bed. This way the slicer was tricked into generating supports which were easier to remove than the raft. If printed directly onto the raft, the model and raft would fuse, which should be avoided.

### 4.11.2 Geometric design

The main task of the frame is carrying the motors, electronics and battery; it should ideally never be responsible for absorbing the shock from an impact. To achieve these goals a design where the hull absorbs the impact from every direction is required. If the hull would fulfill the requirement, the motor arms could be allowed a thinner design, reducing airflow disruption from the propeller downwash. The starting point for the frame and hull design was the 2022 flight controller PCB assembly[50].

As the one of the future plans for the drone is more large scale production, the design aimed to give the frame a shape that can be cut from a single sheet of a given material. Since the drone already had to be modified to accommodate the fiducial marker, further efforts were made to reduce the assembly complexity by reducing the amount of parts required. The polypropylene material making up the drone hull is a soft, pliable plastic; thus in order for the printed hull to be stiff, rigidity was obtained through geometry instead of relying on material properties. A complete seal around the edge of the electronics compartment is required for the structure to be waterproof, although considering the arid conditions inside the drone arena, this attribute is more appropriate for future versions of the drone. Certain limitations were imposed upon the design by the physical arena structure. If the propguards bottom were placed too low, they would collide with the edge of the arena's center hole. Conversely, the drone could not be too tall either. If the center of mass was raised too much, the airframe would not be able to self-right itself after landing on the arena floor.

A predetermined width was set by the size of the QI-charging coil. The vector formed by the CoG in the body oriented reference frame centered at the pivot point was rotated with a  $2 \times 2$  rotation matrix in order to find the location of the CoG when the drone was tilted in the same direction as the metal plates in the arena floor. A positive x value would mean the current design had a built in self righting feature as the CoG would yield a moment rotating it against the metal plate's angle.

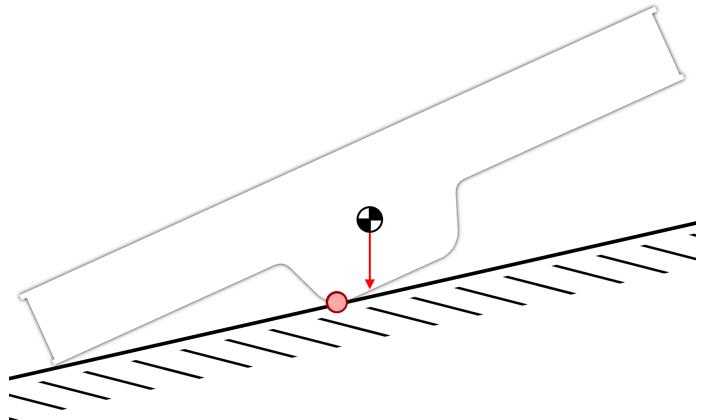


Figure 4.30: Figure illustrating the pivot point

#### 4.11.3 Computational Fluid Dynamics (CFD) Simulation

Using the flow simulation wizard, a new project was created. The analysis type was external, and excluded cavities without flow conditions and internal spaces. Amongst physical features, rotation was selected. Air was the only gas that was added to the system, then the rest of the project wizard was left to default settings. The propellers were selected as rotating regions. To define a rotating region, a part was drawn that covered the entire propeller without intersecting the hull structure. The process was repeated for every rotating body, and the rotating regions were given a rotational speed of 2900 [rad/sec] in CW and CCW direction depending on the placement. To prevent torque generated by the force normal to the rotational movement about the drone's z-axis (extending out from the top of the body), the rotational direction for the propellers were set so half were CW and the other half was CCW, with the same direction across the diagonal. Under the goals menu, a global goal (GG) selecting the force parameter along the lift axis was inserted. Additionally, surface goals (SG) were created for the four propellers. Force, normal force along the lift axis and torque about the lift axis and were selected parameters for the surface goals. It was crucial that the stop criterion for the convergence solver was set to "All Satisfied" in the "Solve and Calculation Control Options" to continue until the values converged fully, and not just a single condition check [93]. The goals criteria selected were for the GG Force, SG Normal Force, and SG Force.

In order to isolate the sources of error from simulations and streamline the process, a simplified set of simulations were done with only a single rotating propeller without the rest of the drone. This small-scale simulation was done multiple times, increasing the mesh refinement until results converged towards similar values. Further complexity was not done, as it would greatly increase solve time with little extra information.

A parametric study was done to simulate multiple motor speeds in sequence. Under "Simulation Parameters" the rotating regions were added. Once added, they were combined into a single study. Right clicking the Values column and editing variation brought up the discrete values table. The maximum motor speed was found by multiplying the 2s battery's maximum voltage with the motor  $K_V$  rating:

$$\omega_{max} = K_V \cdot V = 4500 \left[ \frac{RPM}{V} \right] \cdot 8.4[V] = 37800 \left[ \frac{rev}{min} \right] \simeq 3958 \left[ \frac{rad}{sec} \right] \quad (4.10)$$

A table was provided in SolidWorks with four columns corresponding to each propeller; each row was assigned values from 2900 to 3800 [rad/sec] with increments of 100 [rad/sec], to investigate only motor speeds required for hovering and above. Output parameters for this were the same as the previously inserted calculation goals.

Table 4.5: Propeller Mesh Settings [91]

Setting	Value
Refining cells level	3
Maximum channel refinement level	1
Small solid feature refinement level	3
Curvature level	1
Tolerance level	1

Table 4.6: Global Mesh Settings

Setting	Value
Type	Automatic
Mesh level	5
Minimum gap size	Disabled
Ratio factor	1
Uniform mesh	Disabled
Advanced channel refinement	Disabled
Show basic mesh	Disabled
Close thin slots	Disabled

#### 4.11.4 Impact Test Simulations

Due to complex geometry, the bottom part of the drone hull needed extra fine meshing in order for the drop tests to compute. Mesh control was applied and a 3.221 [mm] mesh with a 1.4 ratio. The remainder of the airframe received a 8.58 [mm] standard mesh with 0.42 [mm] tolerance. Contact surfaces were bonded as the drone will be glued during assembly. Since the hull is composed of a ductile material, the simulation time was increased from 300 microseconds to 5000 microseconds. This was to allow the forces of the impact to propagate through the body, allowing maximum stress and deformation to occur before the end of simulation. An overview of the simulated impact scenarios are shown in table 4.7.

Table 4.7: Impact test scenarios

Name	Description
Flat Fall	1.5 meter fall onto flat plane
Tilted Fall 1	1.5 meter fall landing with 45° rotation onto arena floor - 1 propeller guards impact
Tilted Fall 2	1.5 meter fall landing with 45° rotation onto arena floor - 2 propeller guard impact
Vertical Fall	1.5 meter fall landing with 90° rotation onto arena floor - 1 propeller guard impact

#### 4.11.5 Physical Tests

The drone was assembled with the frame, top and bottom hull, motors and propellers and a battery. The three latter components were selected due to being either the heaviest or most exposed parts. The drone weight with these components attached was 206.1[g]. Prior to any physical impact tests, the drone was placed inside the arena on the metal plates to check the gliding and pivot abilities.

After assembly, the drone was glued together and dropped from the top of the arena with the same orientations and heights as in the simulations. After verifying no damage was sustained, further testing was done from heights exceeding that of the arena. The drone was dropped from increasing heights until the airframe sustained permanent damage to explore the capabilities of the mechanical design.

*This page intentionally left blank.*

# 5 Results

## 5.1 Intel RealSense D435

### 5.1.1 Camera Information

When the software development kit was finished building, the suggested command was sent to the terminal. It gave a long list of information, where the most relevant parts are shown below.

```
Device info:  
  Name : Intel RealSense D435  
  Firmware Version : 5.15.1  
  Product Id : 0B07  
  Usb Type Descriptor : 3.2
```

The USB type descriptor was listed as a 3.X version. The next output was a long list of the depth camera modules supported profiles, which is heavily trimmed down to relevant resolutions and fps.

```
Stream Profiles supported by Stereo Module  
Supported modes:  
  stream   resolution   fps   format  
  Infrared 1 1280x800 @ 25Hz Y16  
  Infrared 1 848x480 @ 90Hz Y8  
  Infrared 1 848x480 @ 60Hz Y8  
  Infrared 2 1280x800 @ 25Hz Y16  
  Infrared 2 848x480 @ 90Hz Y8  
  Infrared 2 848x480 @ 60Hz Y8  
  Depth    1280x720 @ 30Hz Z16  
  Depth    848x480 @ 90Hz Z16  
  Depth    848x480 @ 60Hz Z16
```

Similarly, the list of profiles for the RGB module was also trimmed down.

```
Stream Profiles supported by RGB Camera  
Supported modes:  
  stream   resolution   fps   format  
  Color    1920x1080 @ 30Hz RGB8  
  Color    1920x1080 @ 30Hz Y16  
  Color    1920x1080 @ 30Hz Y8  
  Color    1920x1080 @ 30Hz BGRA8  
  Color    1920x1080 @ 30Hz RGBA8  
  Color    1920x1080 @ 30Hz BGR8  
  Color    1920x1080 @ 30Hz YUYV  
  Color    848x480  @ 60Hz RGB8  
  Color    848x480  @ 60Hz Y16  
  Color    848x480  @ 60Hz Y8  
  Color    848x480  @ 60Hz BGRA8  
  Color    848x480  @ 60Hz RGBA8  
  Color    848x480  @ 60Hz BGR8  
  Color    848x480  @ 60Hz YUYV
```

Finally, the distortion models were listed along the rest of the intrinsic parameters. For comparison, the listings were documented for the same resolution (480p).

```
Intrinsic of "Color" / 848x480 / {YUYV/RGB8/BGR8/RGBA8/BGRA8/Y8/Y16}
Width:      848
Height:     480
PPX:        429.841949462891
PPY:        237.866897583008
Fx:         613.037048339844
Fy:         612.738342285156
Distortion: Inverse Brown Conrady
Coeffs:    0 0 0 0 0
FOV (deg): 69.33 x 42.78

Intrinsic of "Depth" / 848x480 / {Z16}
Width:      848
Height:     480
PPX:        422.719360351562
PPY:        239.679824829102
Fx:         423.671051025391
Fy:         423.671051025391
Distortion: Brown Conrady
Coeffs:    0 0 0 0 0
FOV (deg): 90.04 x 59.06

Intrinsic of "Infrared 1" / 848x480 / {Y8}
Width:      848
Height:     480
PPX:        422.719360351562
PPY:        239.679824829102
Fx:         423.671051025391
Fy:         423.671051025391
Distortion: Brown Conrady
Coeffs:    0 0 0 0 0
FOV (deg): 90.04 x 59.06

Intrinsic of "Infrared 2" / 848x480 / {Y8}
Width:      848
Height:     480
PPX:        422.719360351562
PPY:        239.679824829102
Fx:         423.671051025391
Fy:         423.671051025391
Distortion: Brown Conrady
Coeffs:    0 0 0 0 0
FOV (deg): 90.04 x 59.06
```

The enumerate devices command also provided both the rotation matrices and translation vectors for going between the modules. Only the conversion between the color and depth modules will be presented here. From depth to color, the rotation and translation are respectively:

$$C_{depth}^{color} = \begin{bmatrix} 0.999971 & 0.00677819 & 0.0033387 \\ -0.00676201 & 0.999965 & -0.00483297 \\ -0.00337134 & 0.00481026 & 0.999983 \end{bmatrix}, t_{depth}^{color} = \begin{bmatrix} 0.0150450598448515 \\ -0.000201416929485276 \\ -0.000189087935723364 \end{bmatrix} \quad (5.1)$$

and the inverse operations:

$$C_{color}^{depth} = \begin{bmatrix} 0.999971 & -0.00676201 & -0.00337134 \\ 0.00677819 & 0.999965 & 0.00481026 \\ 0.0033387 & -0.00483297 & 0.999983 \end{bmatrix}, t_{color}^{depth} = \begin{bmatrix} -0.0150466291233897 \\ 0.000100341341749299 \\ 0.00013788026990369 \end{bmatrix} \quad (5.2)$$

### 5.1.2 Camera readings

#### Color Stream

When testing with the different modules, it was experienced difficulty with the various formats. The working formats for the RGB camera with colors were `bgr8`, `rgb8`, `bgra8` and `rgba8`. They provided different color streams, shown in figures 5.1 to 5.4.



Figure 5.1: Screenshot with format: `bgra8`,  
Resolution: 1920x1080



Figure 5.2: Screenshot with format: `bgr8`,  
Resolution: 1920x1080

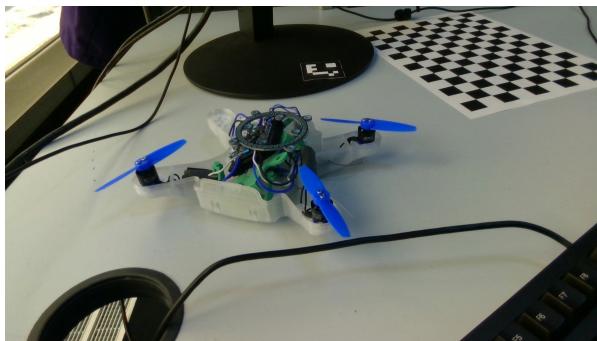


Figure 5.3: Format: `rgba8`,  
Resolution: 1920x1080

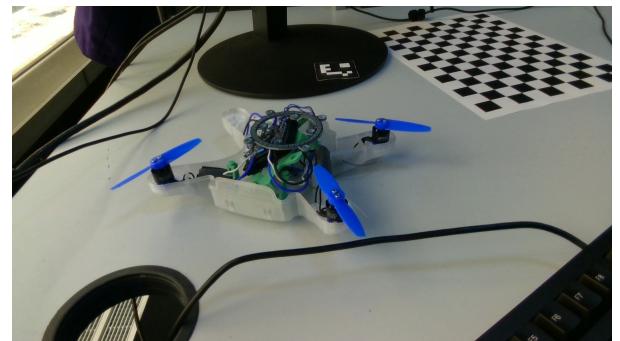


Figure 5.4: Format: `rgb8`,  
Resolution: 1920x1080

The `y8`, `y16` and `yuyv` formats resulted in a greyscale stream. Only the `y8` stream was possible to capture through the script, so the `y16` and `yuyv` frames were saved directly through the button in the display window. The resulting images from these formats are respectively shown in figures 5.5, 5.6 and 5.7.

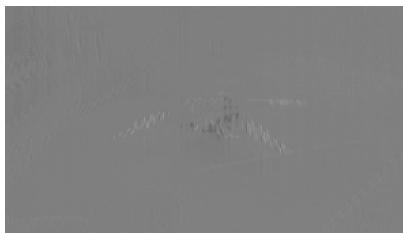


Figure 5.5: Format: `y8`,  
Resolution: 848x480

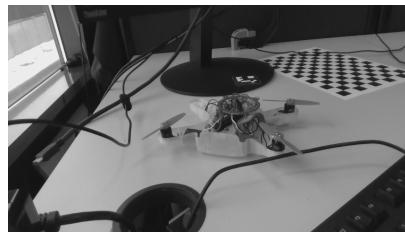


Figure 5.6: Format: `y16`,  
Resolution: 848x480

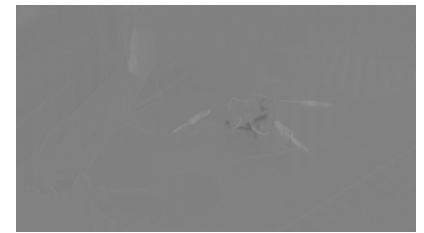


Figure 5.7: Format: `yuyv`,  
Resolution: 848x480

## Infrared Stream

The infrared streams were only available in the `y8` and `y16` formats. The same issue was faced here when exporting the `y16` frame, which here also had to be done directly through the display windows button. When blocking the modules of the camera, it was observed that the image captured from the infrared stream was the left depth camera.



Figure 5.8: Format: `y8`, Resolution: 848x480



Figure 5.9: Format: `y16`, Resolution: 1280x800

## Depth Stream

The depth stream was only captured with the `z16` format in 480p (848x480 pixels), but displayed with different color mappings. The display window had a much darker frame (figure 5.10) than the frame exported through the script (figure 5.11). The frame with the applied colormap was done with `alpha = 0.3`. A smaller alpha resulted with a more dynamic color mapping in short distances, whilst a higher alpha resulted in more dynamic color mapping for longer distances. Above 1 the various `alpha` values gave very dark red images, hardly distinguishable from another.



Figure 5.10: No colormap, exported through display window button



Figure 5.11: No colormap, exported through pressing "s" with the script

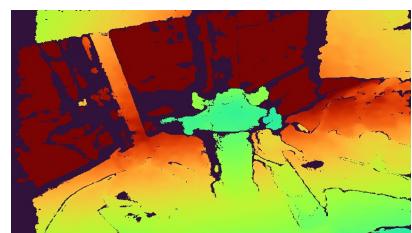


Figure 5.12: Turbo colormap, exported through pressing "s" with the script

## Final Configurations

There were no difficulties with any of the configurations as long as they were set as specified in table 4-2 in the datasheet [24]. Other formats were also tested from the `pyrealsense2` library, but they resulted with a runtime error ("Couldn't resolve requests"). The final configuration for the modules were set to be the following:

Table 5.1: Camera Module Configurations

Camera Module	Resolution [pixels]	Format	fps
RGB Camera	848x480	bgr8	60
Depth Camera	848x480	z16	60
Infrared Projector	848x480	y8	60

## 5.2 Machine Vision

### 5.2.1 Marker Detection

The marker detection was done with multiple markers, but the actual test results with the different versions will be displayed in the subsequent sections. A frame was captured while streaming the tracking of a marker, shown in figure 5.13. The marker for this test was ArUco with a 6x6 dictionary. The displayed frame outline and ID were updated each frame while streaming, as their methods and algorithms are located inside the streams loop.



Figure 5.13: Screenshot taken of marker detection.

### 5.2.2 Intrinsic Parameters

The intrinsics extracted from the camera using CustomRW in the terminal yielded the following (relevant) results:

```

CustomRW for Intel RealSense D400, Version: 2.13.1.0

Device name: Intel RealSense D435
Firmware version: 05.15.01.00

Calibration parameters from the device:
resolutionLeftRight: 1280 800

FocalLengthLeft: 636.503845 635.518372
PrincipalPointLeft: 637.250732 400.241577
DistortionLeft: -0.056916 0.063536 0.000103 0.000426 -0.020005

FocalLengthRight: 636.887390 635.947205
PrincipalPointRight: 634.166443 398.218475
DistortionRight: -0.056490 0.061075 0.000289 0.000054 -0.018888

RotationLeftRight: 0.999944 -0.003210 -0.010073
                   0.003198 0.999994 -0.001213
                   0.010077 0.001181 0.999949
TranslationLeftRight: -50.033840 -0.040516 -0.189608

resolutionRGB: 1920 1080

FocalLengthColor: 1379.333496 1378.661255
PrincipalPointColor: 973.144470 535.200500
DistortionColor: 0.000000 0.000000 0.000000 0.000000 0.000000
RotationLeftColor: 0.999986 0.004384 -0.002950
                  -0.004400 0.999976 -0.005397
                  0.002926 0.005410 0.999981
TranslationLeftColor: 15.045058 -0.201417 -0.189088

```

As shown in the listing above, this yielded only the parameters for the maximum resolution for each module. From this two arrays were created in Python using the `numpy` library. The camera matrix was created as in (2.7) with the focal lengths and optical center points.<sup>1</sup> The parameters extracted from this are given in pixels [27].

$$\text{FocalLengthColor: } 1379.333496, \quad 1378.661255 = (f_x, f_y) \quad (5.3)$$

$$\text{PrincipalPointColor: } 973.144470, \quad 535.200500 = (c_x, c_y) \quad (5.4)$$

Subsequently, the distortion coefficients (which were set to 0 for all coefficients) were implemented as another array:

$$\text{distortionCoefficients} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \quad (5.5)$$

### 5.2.3 Camera Calibration

#### D400 Dynamic Calibration Tool

The camera calibration done with the D400 Dynamic Calibration Tool was done with a distance of 70 centimeters away from the camera as suggested in the guide [120] with the calibration figure printed on a A4 paper. The calibration window is displayed in figure 5.14.

---

<sup>1</sup>Assuming the optical center has the same x and y coordinates as the principal point, due to them both being coincident with the optical axis.

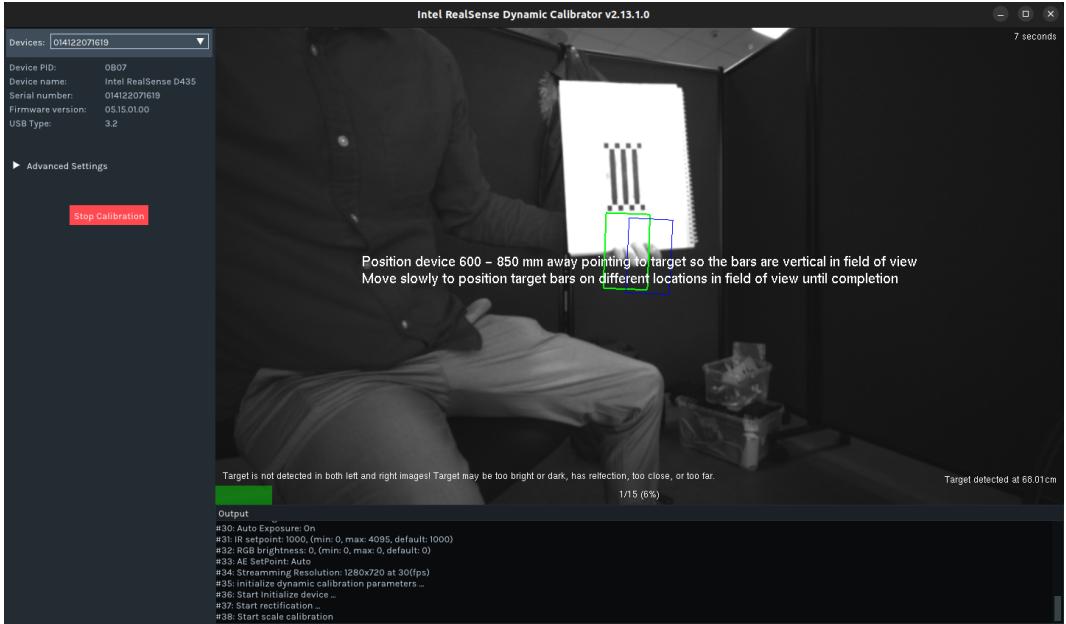


Figure 5.14: D400 Dynamic Calibration Tool

The resulting distortion coefficients were still all zero, with the camera matrix for the 480p resolution extracted from the SDK like before:

$$\text{cameraMatrixSDK} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 613.03705 & 0.0 & 429.8419 \\ 0.0 & 612.7383 & 237.8669 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (5.6)$$

## OpenCV Calibration

A single image from the imported folder was displayed with and without the cropping, shown respectively in figure 5.15 and 5.16.

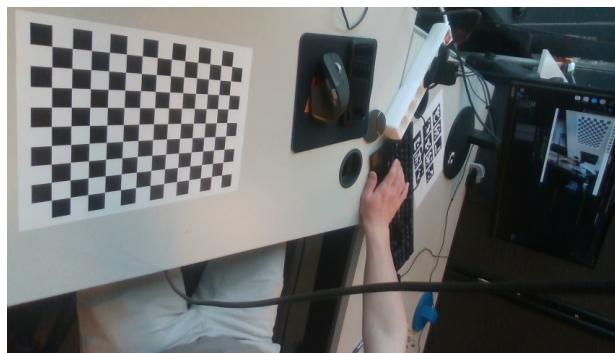


Figure 5.15: Original image without undistortion.

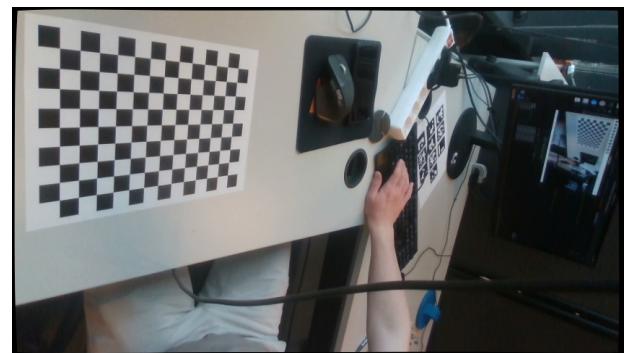


Figure 5.16: Image with undistortion from calibration.

The camera calibration done in OpenCV gave both results for the camera matrix and the distortion coefficients:

$$\text{cameraMatrixOpenCV} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 608.7630 & 0.0 & 439.3740 \\ 0.0 & 609.2398 & 232.7132 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (5.7)$$

$$\text{distCoeffsOpenCV} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \quad (5.8)$$

$$= \begin{bmatrix} 0.21043186 & -0.69360305 & -0.00180299 & 0.00226683 & 0.65082012 \end{bmatrix} \quad (5.9)$$

### 5.2.4 Manual Measurement Test

To combat poor detection at longer distances, the physical size of the marker was increased from a 40 [mm] to 50 [mm], as well as reducing the dictionary from 7x7 to 6x6. The distances were measured extensively in z-direction, by trying many different configurations of the camera matrices and distortion vectors as well as multiple calibrations, but the final results are shown in tables 5.2 and 5.3.

Table 5.2: Measured and estimated z-distances: SDK Matrix Combinations

Combination	Measured (Mean) [m]	Estimated [m]	Error [m]
SDK Matrix & SDK Coeffs	1.60	1.6528	0.0528
	1.20	1.2227	0.0227
	0.40	0.3945	-0.0055
SDK Matrix & OpenCV Coeffs	1.60	1.6670	0.0670
	1.20	1.2266	0.0266
	0.40	0.4038	0.0038

Table 5.3: Measured and estimated z-distances: OpenCV Matrix Combinations

Combination	Measured (Mean) [m]	Estimated [m]	Error [m]
OpenCV Matrix & OpenCV Coeffs	1.60	1.5998	-0.0002
	1.20	1.2190	0.0190
	0.40	0.3996	0.0004
OpenCV Matrix & SDK Coeffs	1.60	1.6047	0.0047
	1.20	1.2057	0.0057
	0.40	0.3973	-0.0027

After the z-distances were measured, the x- and y coordinates were verified. To do this, a crosshair was displayed on the image through OpenCV's `line()` method, shown in figure 5.17.

It was realized that the focal lengths gave a more correct scale from OpenCV, though not having the best estimate of the optical center. The vertical center  $c_y$  was quite accurate, but the horizontal center  $c_x$  was less correct; therefore it was reduced by 10 pixels, which gave a better estimate. This was inspired by the parameter from the Dynamic Calibration Tool, since it had a better center estimate in the x-direction. The alteration yielded an unnoticeable change in the results for the z-estimate. With this change, the final estimate of the x distance was 14.72 [cm] and the y distance 15.33 [cm] (average of the past 6 values for both), when measured 15 [cm] in x and y direction, around 30 [cm] away in z direction. The x-estimate was even worse when measured at longer distances, which prior to the change was estimated to be 10.96 [cm] and after estimated to be 13.99 [cm] when the camera was about 1.5 [m] away from the marker. The two combinations with the camera matrix from OpenCV shown in table 5.3 are very similar, but the choice was to continue

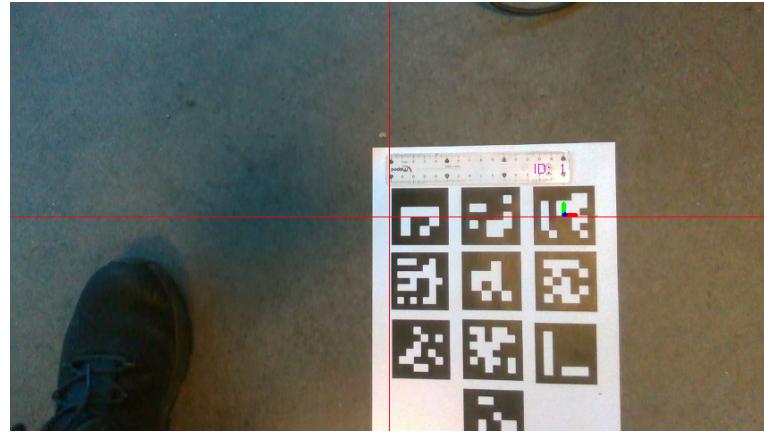


Figure 5.17: Red crosshair display for verifying pose estimate.

with the OpenCV matrix and the SDK coefficients since the distortion in the image was minimal as shown in figure 5.16 (when compared, the average error is also lower with this combination even though both are small). Also less computing power may be used when not having to account for the distortion. Therefore, the final combination was the OpenCV matrix with a modified x center and the SDK coefficients:

$$\text{Camera Matrix} = \begin{bmatrix} 608.7630 & 0.0 & 429.3740 \\ 0.0 & 609.2398 & 232.7132 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (5.10)$$

$$\text{Distortion Coefficients} = [0.0 \ 0.0 \ 0.0 \ 0.0 \ 0.0] \quad (5.11)$$

### 5.2.5 Depth Readings

The on-chip calibration yielded a value of -0.12 (good). The calculated average values for each distance yielded the following results at the measured heights:

Table 5.4: Depth Reading Results 1.6 to 0.9 [m]

Measured [mm]	Estimated Average [mm]
1600	1618.3
1500	1509.3
1400	1416.1
1300	1311.0
1200	1202.4
1100	1107.8
1000	1003.4
900	902.8

Table 5.5: Depth Reading Results 0.8 to 0.1 [m]

Measured [mm]	Estimated Average [mm]
800	800.8
700	703.9
600	602.3
500	500.5
400	400.4
300	300.1
200	201.3
100	NA

### 5.2.6 Combined Marker Pose Estimate and Depth Reading

With both the marker detection and depth reading, the frames were displayed simultaneously. The color stream was set up to display the pose axes in the marker center and the depth stream following the center with the crosshair. This is shown respectively in figure 5.18 and 5.19.

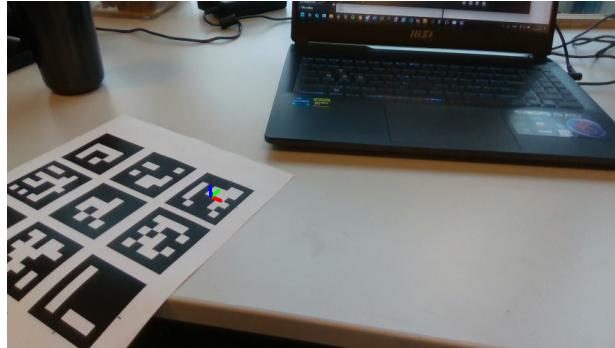


Figure 5.18: Color stream with pose axes

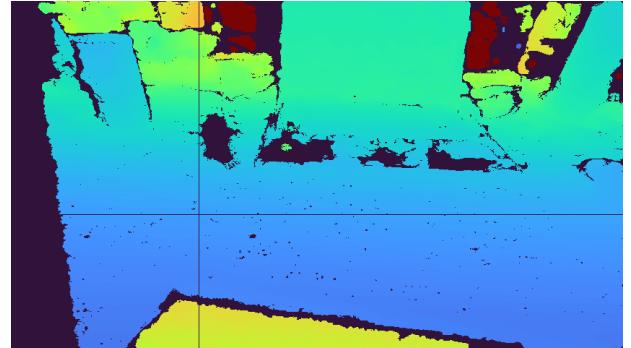


Figure 5.19: Depth stream with crosshair

### 5.2.7 Marker Dictionary Test

The marker dictionary test was done initially outside of the arena by moving the camera around the markers. Later it was done inside the arena with the camera mounted in the bracket atop of the arena, while moving the markers around with different orientations. This was done to get a most dynamic movement possible, all the while recording it for giving the same base for the analysis of each dictionary. The results for the different marker dictionaries outside the arena are shown in table 5.6.

Table 5.6: Marker Test Results: Outside of arena

Dictionary	Computation Time [seconds]	Frames with data
<b>ArUco 4x4 (50 IDs)</b>	3.6940	727
<b>ArUco 5x5 (50 IDs)</b>	3.6855	746
<b>ArUco 6x6 (50 IDs)</b>	3.6719	714
<b>ArUco 7x7 (50 IDs)</b>	3.7207	710
<b>ArUco Original</b>	3.8600	741
<b>AprilTag 16h5</b>	3.5419	643
<b>AprilTag 25h9</b>	3.5153	547
<b>AprilTag 36h10</b>	4.1156	561
<b>AprilTag 36h11</b>	3.7997	495
<b>ArUco MIP 36h12</b>	4.0306	812

The total amount of frames for this recording was 878. The final results for the test inside the arena with the camera mounted are shown in table 5.7.

Table 5.7: Marker Test Results: Inside of arena

Dictionary	Computation Time [seconds]	Frames with data	False Positives
<b>ArUco 4x4 (50 IDs)</b>	6.8086	1327	30+
<b>ArUco 5x5 (50 IDs)</b>	6.7228	1328	2
<b>ArUco 6x6 (50 IDs)</b>	6.8764	1293	0
<b>ArUco 7x7 (50 IDs)</b>	6.9959	1278	0
<b>ArUco Original</b>	7.3110	1354	2
<b>AprilTag 16h5</b>	6.9142	1233	2
<b>AprilTag 25h9</b>	7.1363	1151	0
<b>AprilTag 36h10</b>	7.8651	1069	0
<b>AprilTag 36h11</b>	7.0430	1024	0
<b>ArUco MIP 36h12</b>	7.4596	1431	1000+

This test had a total amount of 1488 frames, and proved the 5x5 ArUco dictionary to be the clear choice, being both the fastest and most stable. It is important to note that the "False Positives" column in the table is an approximation from visual inspection of the exported videos. The only dictionaries with a significant amount of false positives are the ArUco 4x4 and MIP 36h12 dictionaries which also detected the reflection in the arenas transparent panels. The other dictionaries were way more robust against false positives, only detecting incorrectly on the other marker dictionaries. As only one marker will be visible at a time in the final system, this was considered negligible. The exported videos of the recording with the displays are shown in the demonstration video of the project.

### 5.3 Pose Validation with Qualisys

#### 5.3.1 Physical Problems and Solutions

When the group performed the tests, some issues were encountered and addressed, and important details were added to the test plan. First off, to ensure only the small marker on the cube is tracked by the D435 camera, the big marker has to be blocked or removed. When moving the marker, it was important to be aware of both the physical surroundings, but also the tracking on both systems. The D435 camera has a limited FOV to be aware of, but it was also experienced that the motion capture had limitations as well. This is further illustrated in the figures in the subsequent subsection. Though there are many cameras, certain locations in the volume may not be covered and give a poor estimate, not to mention when moving the cube around. Even when being inside the covered and calibrated volume, the motion capture tracking may not be working if the operator's arm is occluding one of the cameras or the reflective balls.

#### 5.3.2 Calibration and Setup

The motion capture calibration done with the Qualisys rod yielded the results shown in table 5.8.

Table 5.8: QTM Calibration Results

Camera	X [mm]	Y [mm]	Z [mm]	Points	Average Residual [mm]
01	244.39	-735.24	834.98	1203	0.43963
02	757.00	842.17	1178.33	652	0.60689
03	836.68	317.17	226.80	880	0.51988
04	-727.08	352.31	294.11	1398	0.43711
05	-485.56	680.86	1312.05	1568	0.53061
06	237.14	-684.31	142.76	1435	0.44619
07	867.51	700.08	136.33	518	0.52705
08	-551.93	723.82	150.46	702	0.52308

The resulting calibrated volume is shown in figure 5.20. To demonstrate the volume covered by the machine vision which tracks the fiducial marker, a frustum was modeled in SolidWorks. It is based on the FoV from the RGB camera in table 2.2, which is  $69^\circ \times 42^\circ$ , and is shown in figure 5.21. It was made with a lofted base between the known rectangle and another with dimensions related through the known dimensions and trigonometry

$$W = \tan\left(\frac{FoV_w}{2}\right) \cdot 2 + w = \tan\left(\frac{69^\circ}{2}\right) \cdot 2 + 2.7288 = 2064.5717[\text{mm}]$$

$$H = \tan\left(\frac{FoV_h}{2}\right) \cdot 2 + h = \tan\left(\frac{42^\circ}{2}\right) \cdot 2 + 1.5498 = 1153.1419[\text{mm}]$$

and placed in the center of the x-axis and offset 32.5 [mm] from the y-axis of the camera in the slot of the black bracket atop the arena, based on the technical drawing from the datasheet [24].

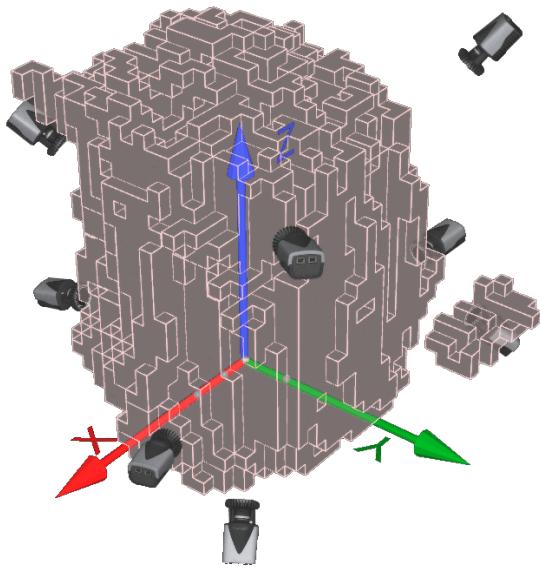


Figure 5.20: Calibrated Voxel Volume from QTM



Figure 5.21: D435 Camera frustum in arena

The transformation between the QTM frame and D435 camera frame was done in the "poseTestSynchExport" script (Appendix G.2.5) with a rotation and a translation each iteration. Rotation was done by multiplication of the matrix shown in (4.1) as a rotation about the y-axis. The translation vector was calculated by subtracting the current vector from the physically measured vector

$$\begin{aligned}\overrightarrow{C_Q D} &= \overrightarrow{Q C} - \overrightarrow{Q D} \\ &= [x_{CQ} \quad y_{CQ} \quad z_{CQ}]\end{aligned}\quad (5.12)$$

and flipping the sign of the y-component

$$\overrightarrow{C D} = [x_{CQ} \quad -y_{CQ} \quad z_{CQ}] \quad (5.13)$$

As shown in figure 5.22, this is done because of the rotation. This is further discussed in section 6.4.2. The sign flip of the vectors y-element was done in the subsequent scripts, since this oversight was realised after all the data was exported.

The pose validation test was filmed from both the motion capture system, the D435 camera and with a personal phone. The video has been edited and is shown in the bachelor video, otherwise an image from the video is shown below.

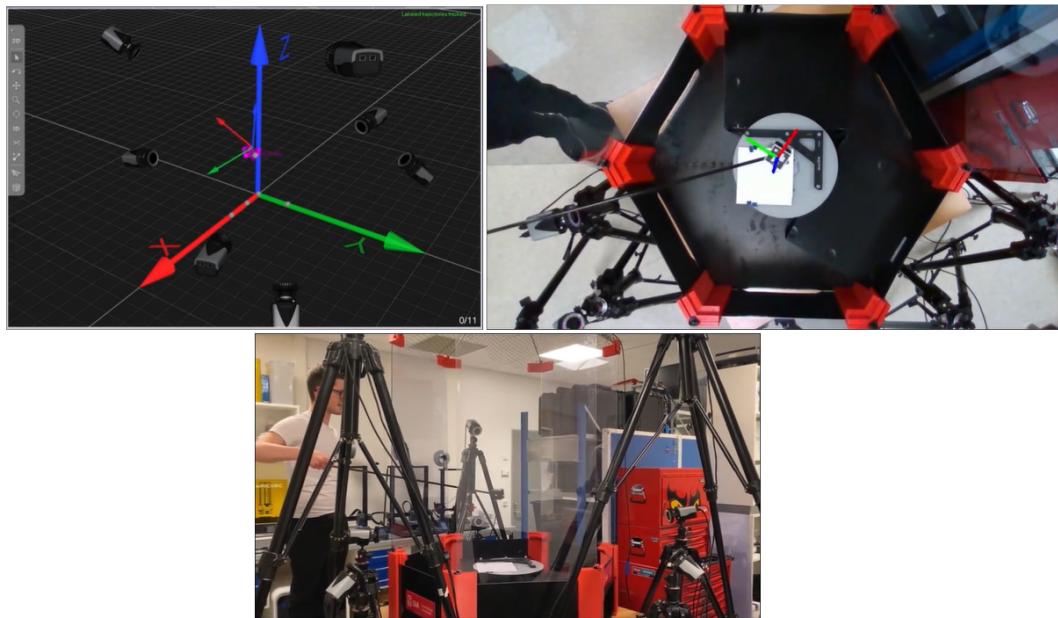


Figure 5.23: Screen capture from video of pose validation test. Top left is QTM tracking, top right is OpenCV tracking and bottom is phone recording

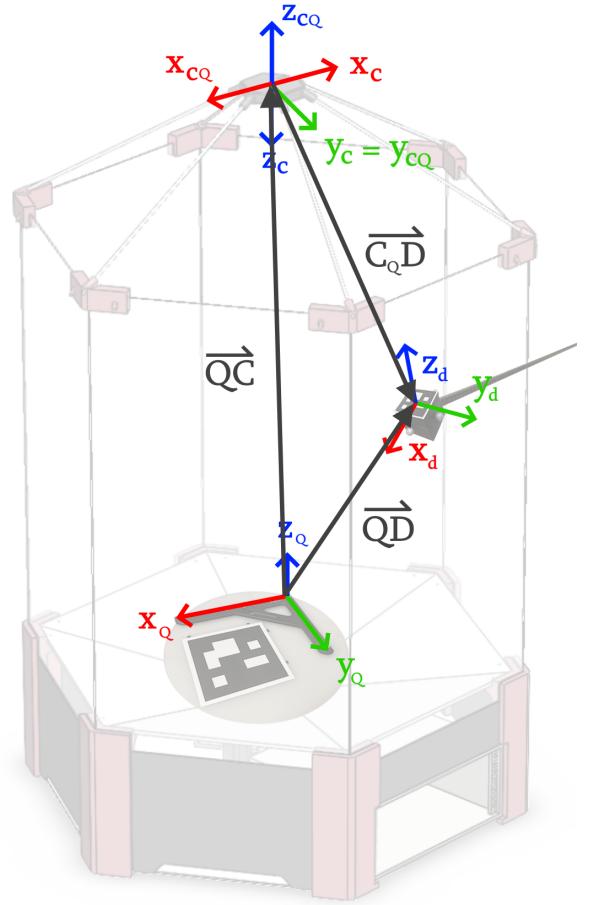


Figure 5.22: Translation between reference frames for QTM and D435

### 5.3.3 Plotting Results

As the scripts made for the test were designed to be relatively modular, there are a lot of figures and data to be presented. To keep this section organized, regions of interest from two of the recorded tests (test #2 and #3) will be shown here. The full plots of all tests are shown in appendix E.3.1, E.3.2, E.3.3, E.3.4 and E.3.5. The data, both raw and analysed, is not included in the appendix, but is found in the GitHub repository. To reiterate, the full overview of scripts, data and their contents are shown in E.2.

Five tests were recorded for this project to give sufficient data. The first two tests (#0 and #1) were primarily focused on the estimated position and not orientation, due to the limited movement in the "windows" of the arena (where the transparent walls had been removed) because of physical obstacles. This was mainly due to the motion capture cameras and a storage shelf being in the way, where the third "window" had much more space to move around freely. The third test (#2) was done in this window, where the marker was kept relatively level with the ground. The fourth test (#3) captured approximately the same volume in the arena, but with much more rotational movement. This was to demonstrate the different properties and performance of the system with the two most distinguishable tests. The final test (#4) was to simulate how the drone may move when being controlled by the RL in the future.

Before presenting the plotting results, the average of the absolute value of the difference  $|\Delta|$  between both translation vectors are shown in table 5.9. Note that some of the XY-projected angles have been wrapped both here and in appendix, an overview is provided in table 5.10 to give a better understanding of the angle values in the plots.

Table 5.9: Average  $|\Delta|$  between Translation Vector Solutions

Test #	x [mm]	y [mm]	z [mm]
Test 0	0.34423	0.21768	0.40127
Test 1	0.24889	0.22608	0.22967
Test 2	0.20082	0.13124	0.19575
Test 3	0.22680	0.24302	0.36219
Test 4	0.13969	0.24691	0.22286

Table 5.10: Wrapping Overview

Test #	xWrapping	yWrapping
0	true	false
1	false	false
2	false	true
3	false	true
4	false	false

### 5.3.3.1 Test #2 Results

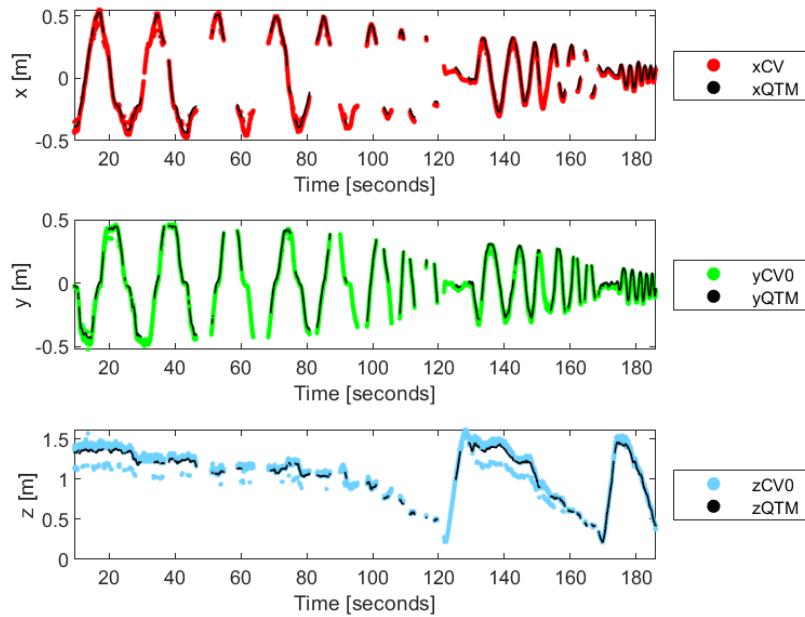


Figure 5.24: Position plot, comparing elements of translation vectors

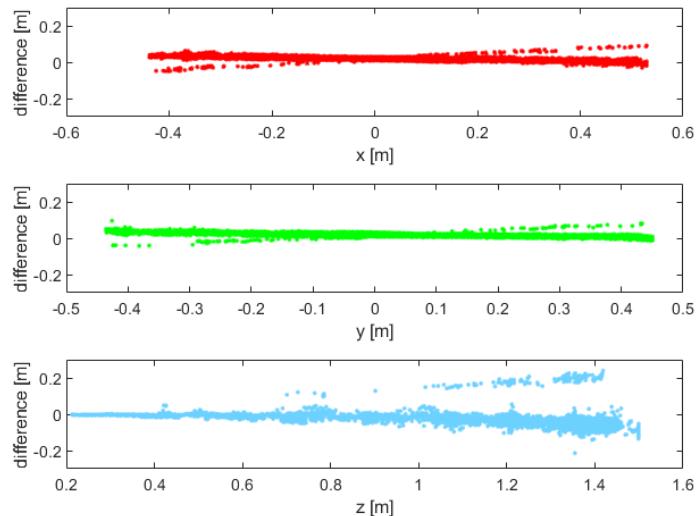


Figure 5.25: Difference between first estimated translation and QTM, as function of QTM position

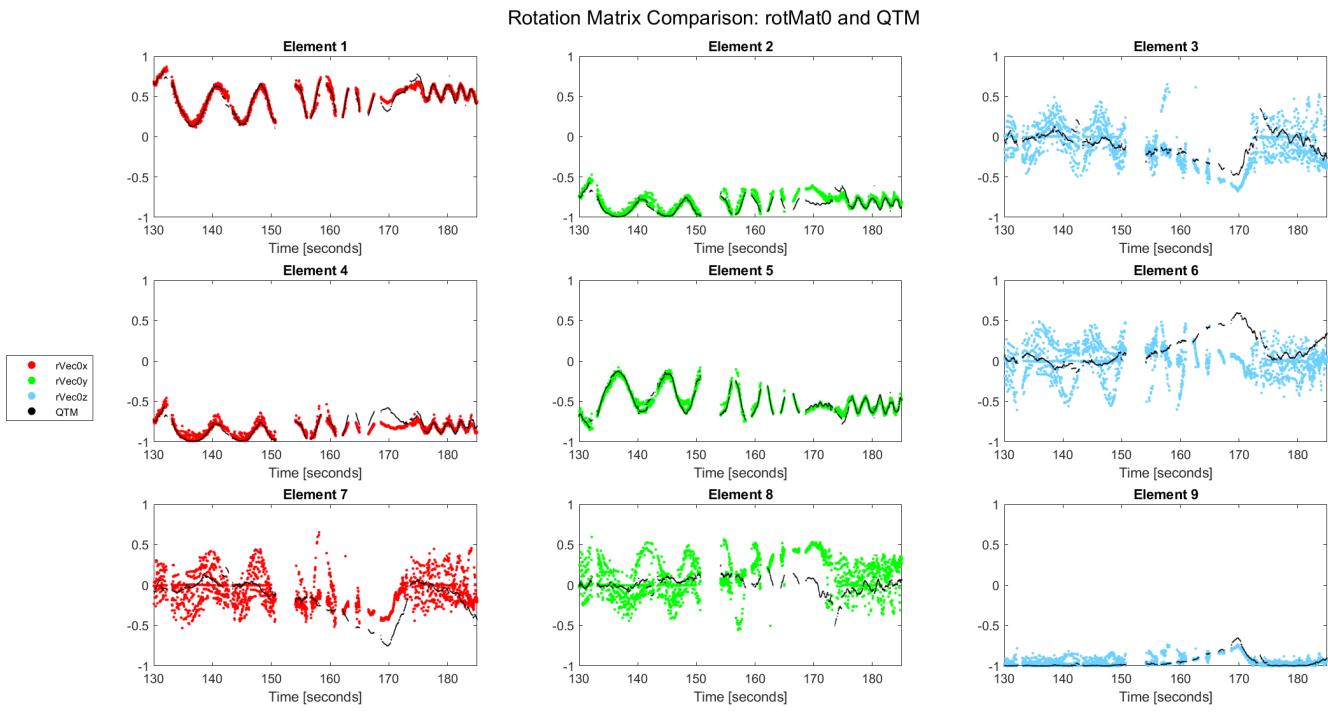


Figure 5.26: Comparison of QTM and first estimate's rotation matrix elements

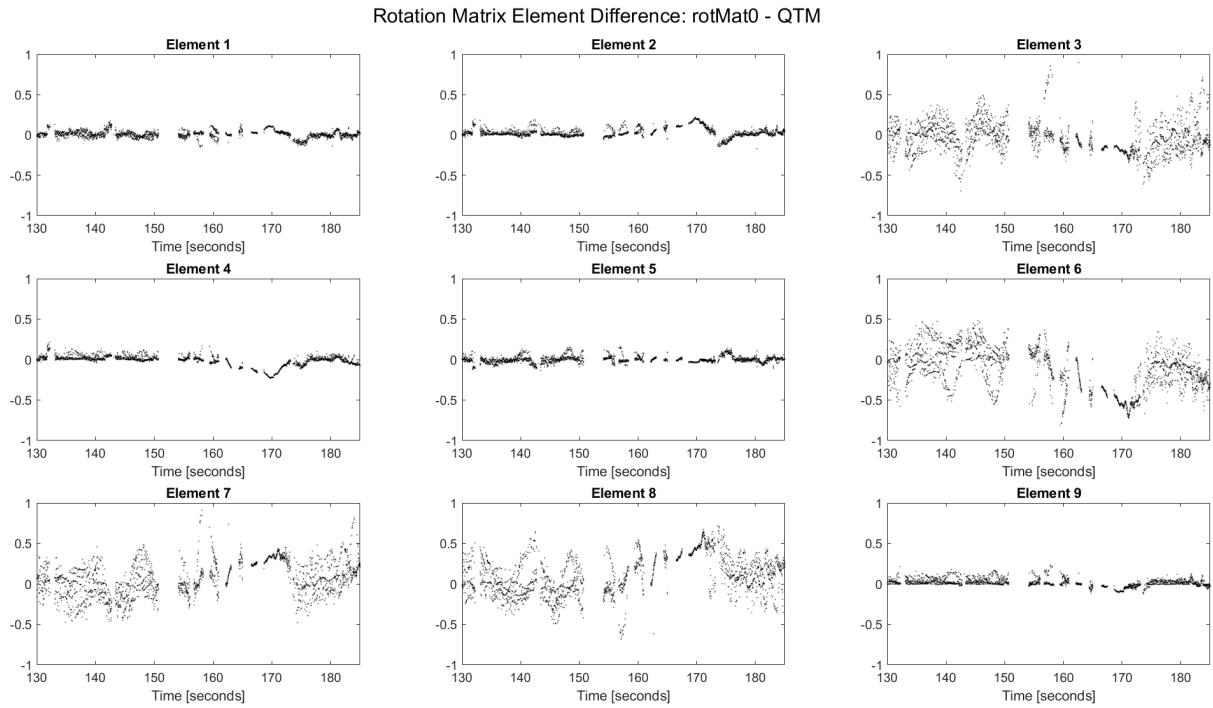


Figure 5.27: Difference between QTM and first estimate's rotation matrix elements

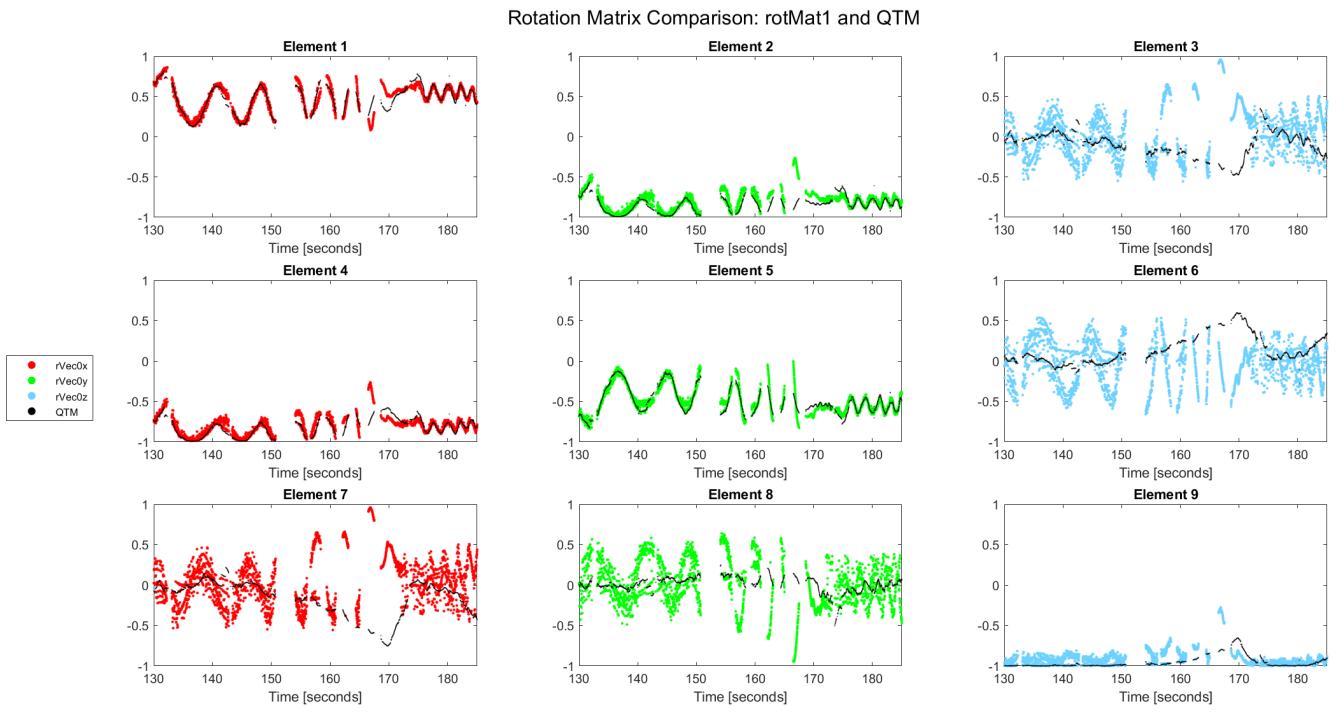


Figure 5.28: Comparison of QTM and second estimate's rotation matrix elements

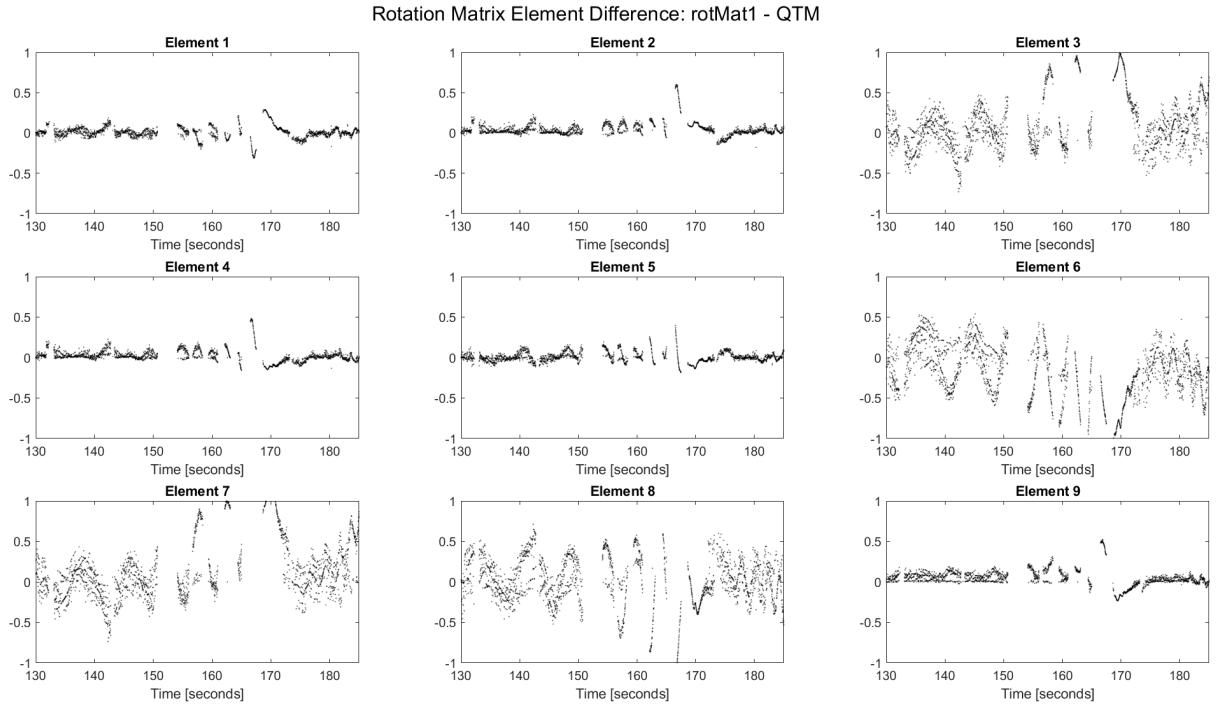


Figure 5.29: Difference between QTM and second estimate's rotation matrix elements

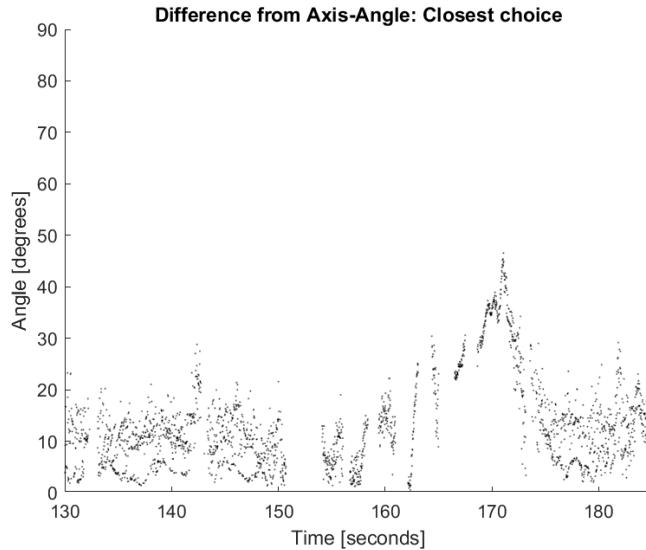


Figure 5.30: Axis-Angle plot of angular difference between QTM and closest estimate

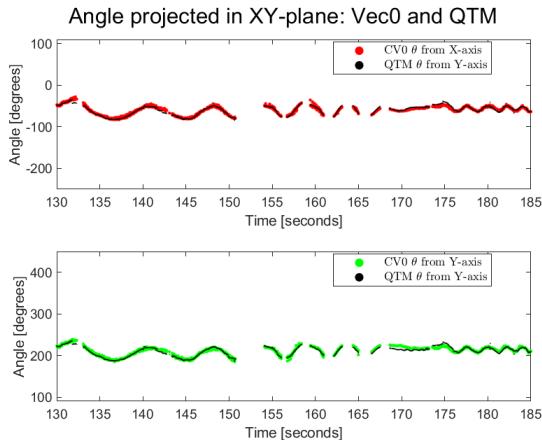


Figure 5.31: XY-projected angle between global and first estimated axes, compared to QTM

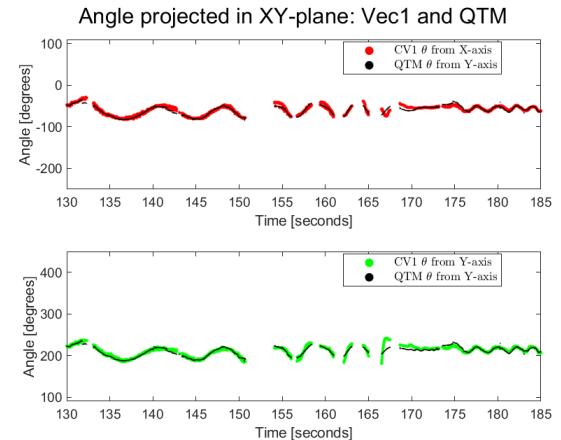


Figure 5.32: XY-projected angle between global and second estimated axes, compared to QTM

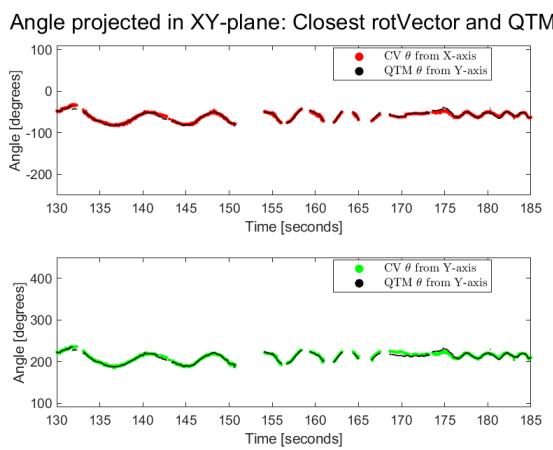


Figure 5.33: XY-projected angle between global and chosen estimated axes, compared to QTM

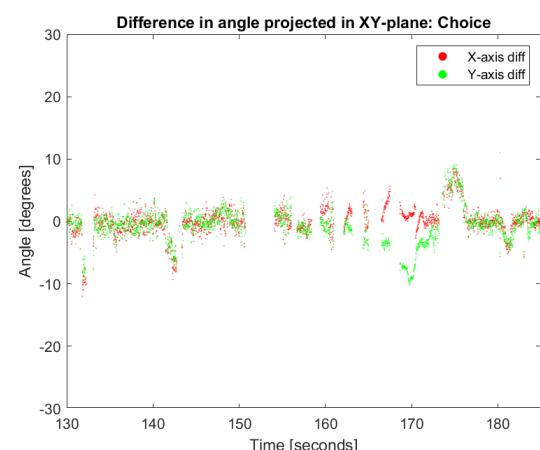


Figure 5.34: XY-projected angle difference between chosen estimate and QTM axes

### 5.3.3.2 Test #3 Results

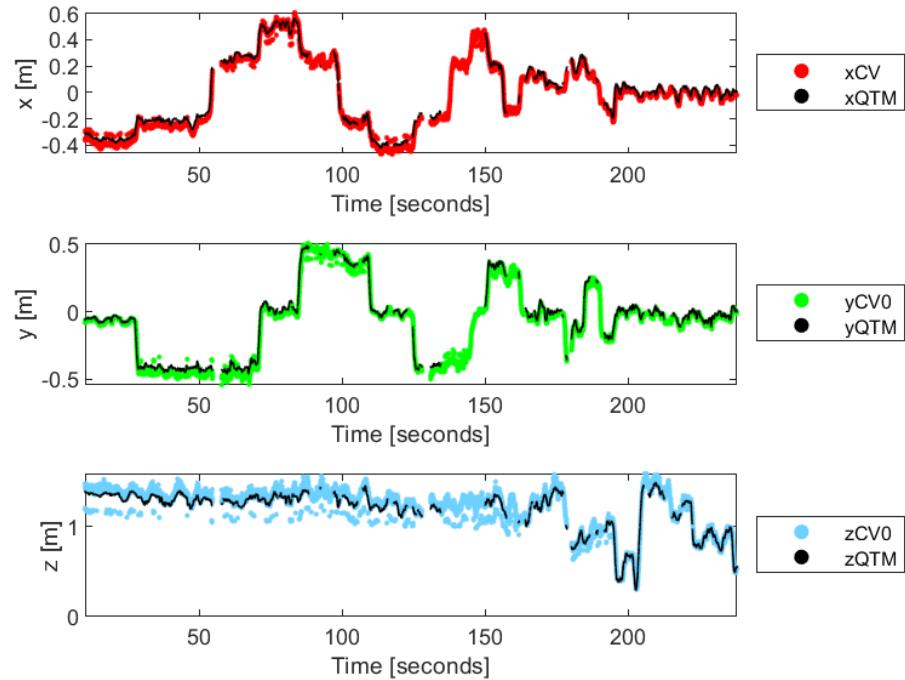


Figure 5.35: Position plot, comparing elements of translation vectors

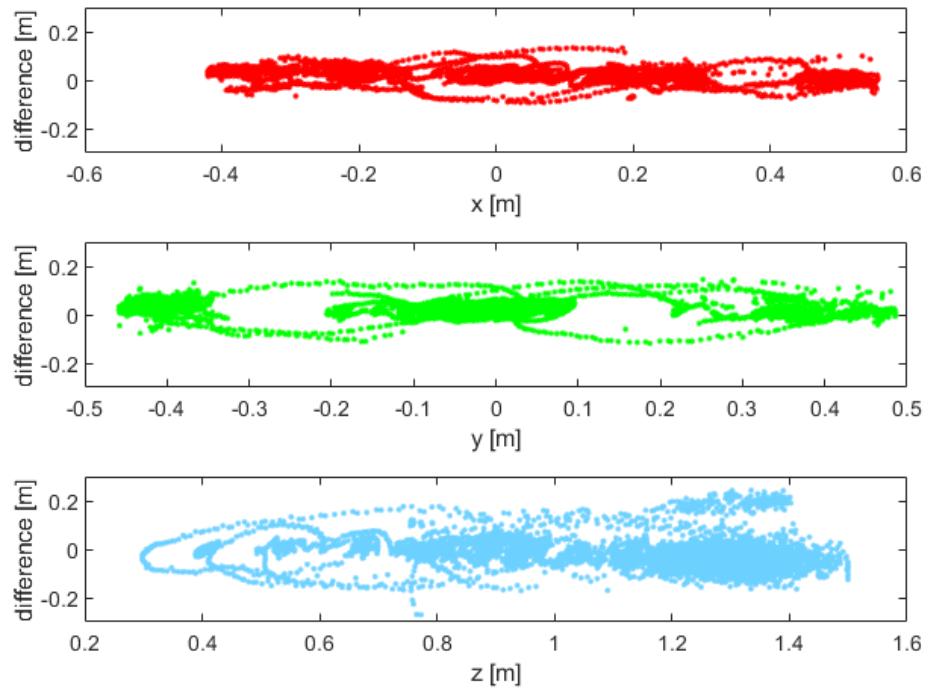


Figure 5.36: Difference between first estimated translation and QTM, as function of QTM position

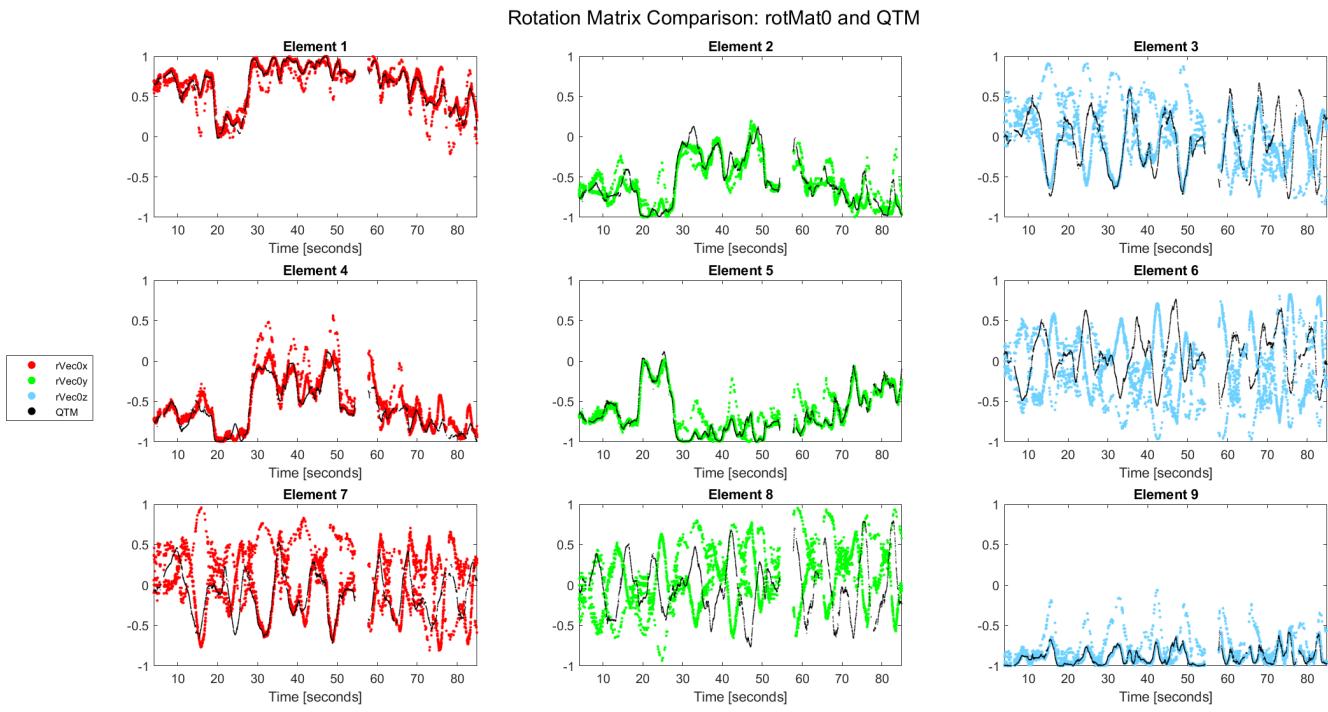


Figure 5.37: Comparison of QTM and first estimate's rotation matrix elements

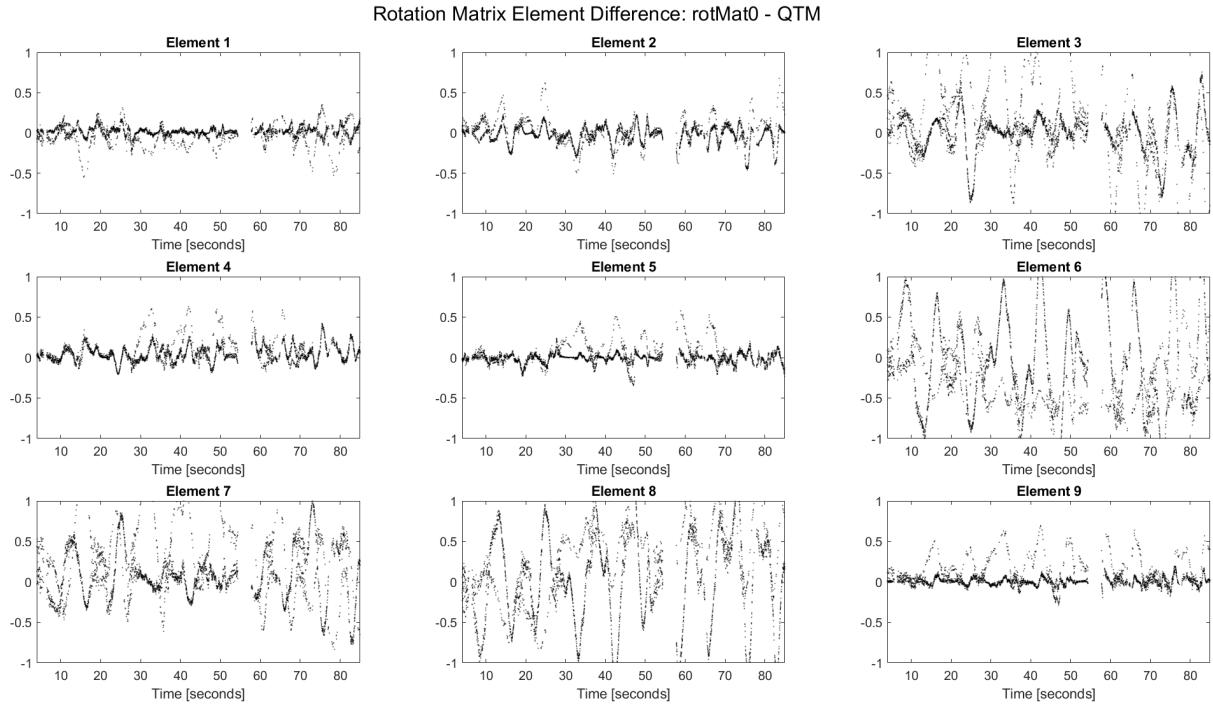


Figure 5.38: Difference between QTM and first estimate's rotation matrix elements

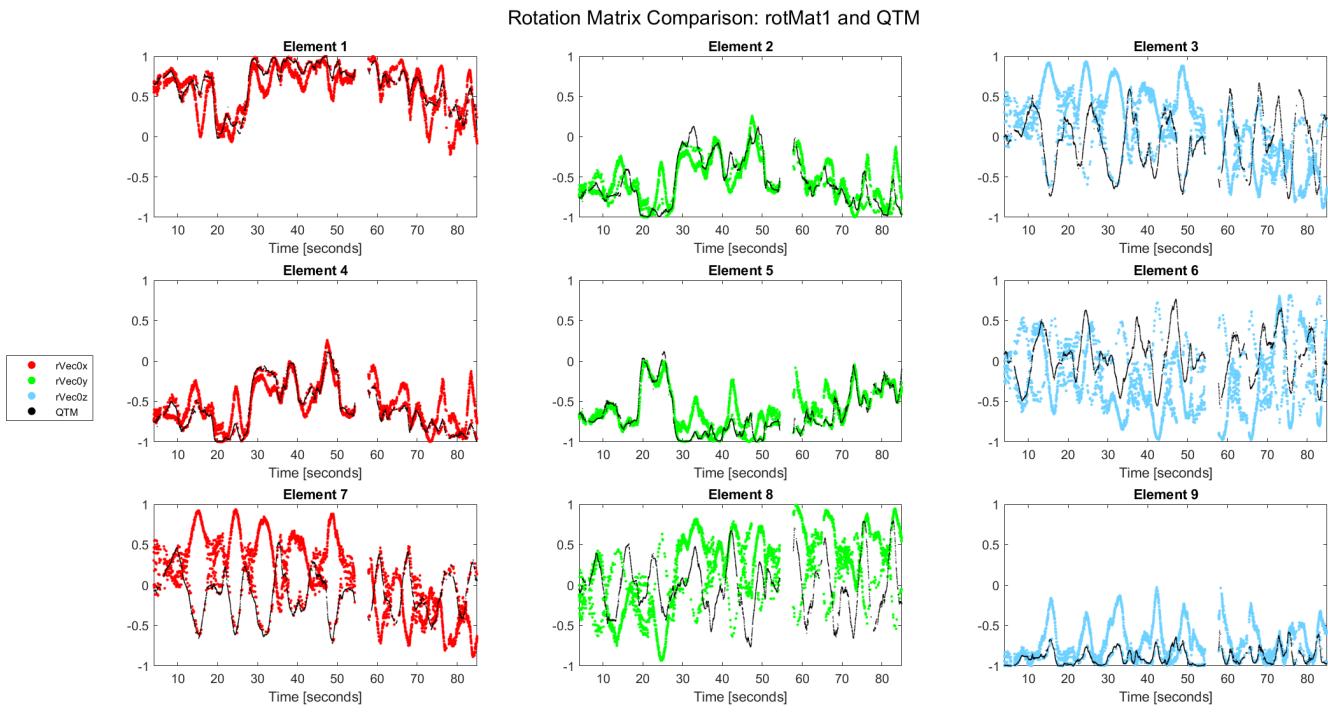


Figure 5.39: Comparison of QTM and second estimate's rotation matrix elements

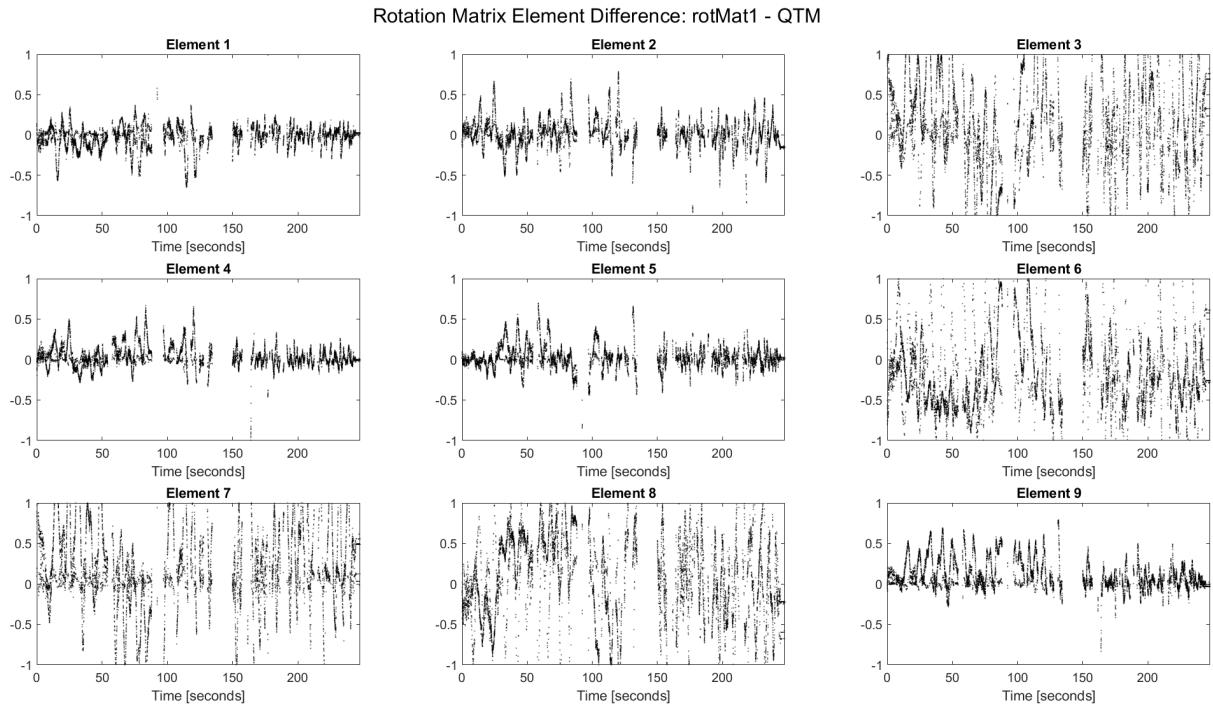


Figure 5.40: Difference between QTM and second estimate's rotation matrix elements

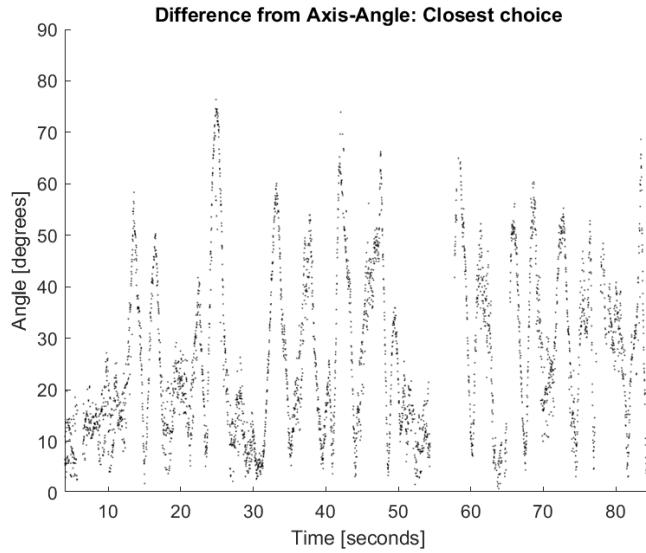


Figure 5.41: Angle-Axis plot of angular difference between QTM and closest estimate

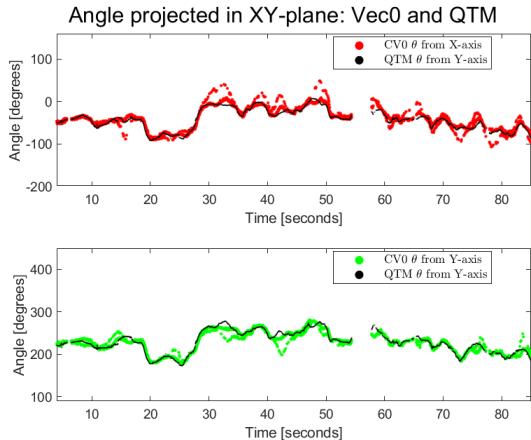


Figure 5.42: XY-projected angle between global and first estimated axes, compared to QTM

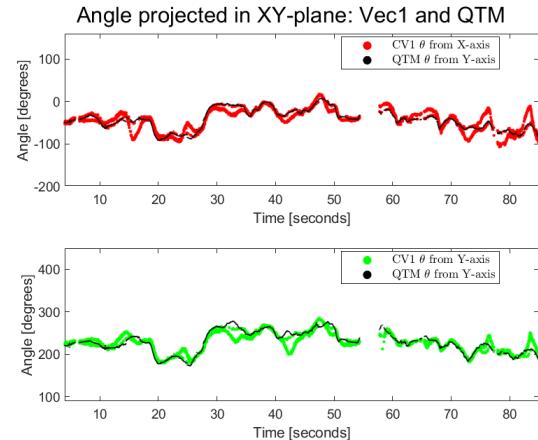


Figure 5.43: XY-projected angle between global and second estimated axes, compared to QTM

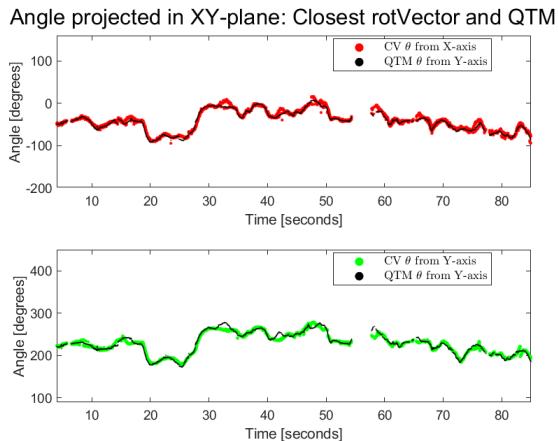


Figure 5.44: XY-projected angle between global and chosen estimated axes, compared to QTM

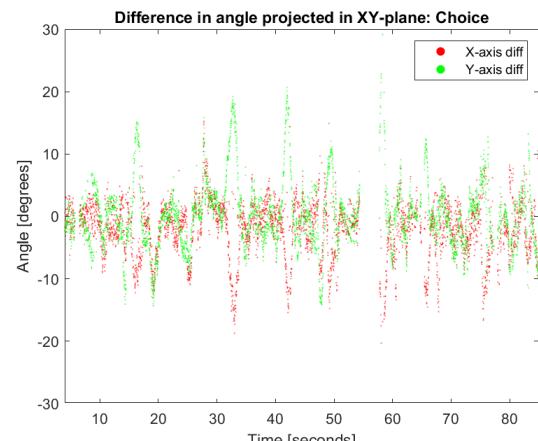


Figure 5.45: XY-projected angle difference between chosen estimate and QTM axes

### 5.3.4 Translation Distribution Fitting

The combined data from tests 0,1,2 and 4 were implemented and used to plot the difference between the estimated positions and QTM, as function of QTM. This is shown in figure 5.46. The corresponding confidence intervals from the distribution fitting are shown for x,y and z in figures 5.47, 5.48 and 5.49 respectively. The results are from dividing them into 9 intervals.

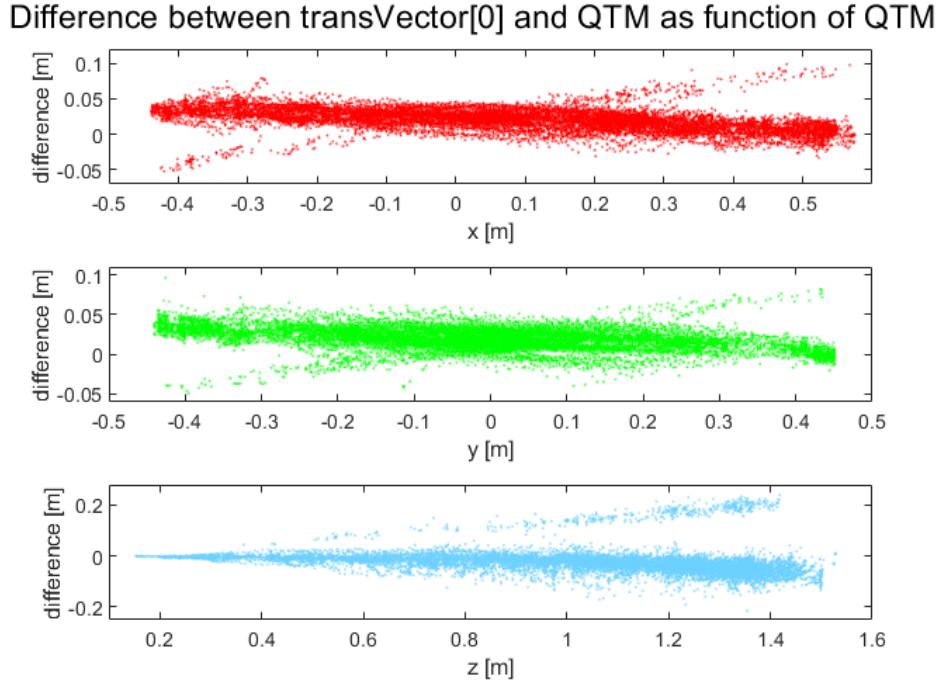


Figure 5.46: Translation difference as function of QTM measurement. Data from test 0,1,2 and 4.

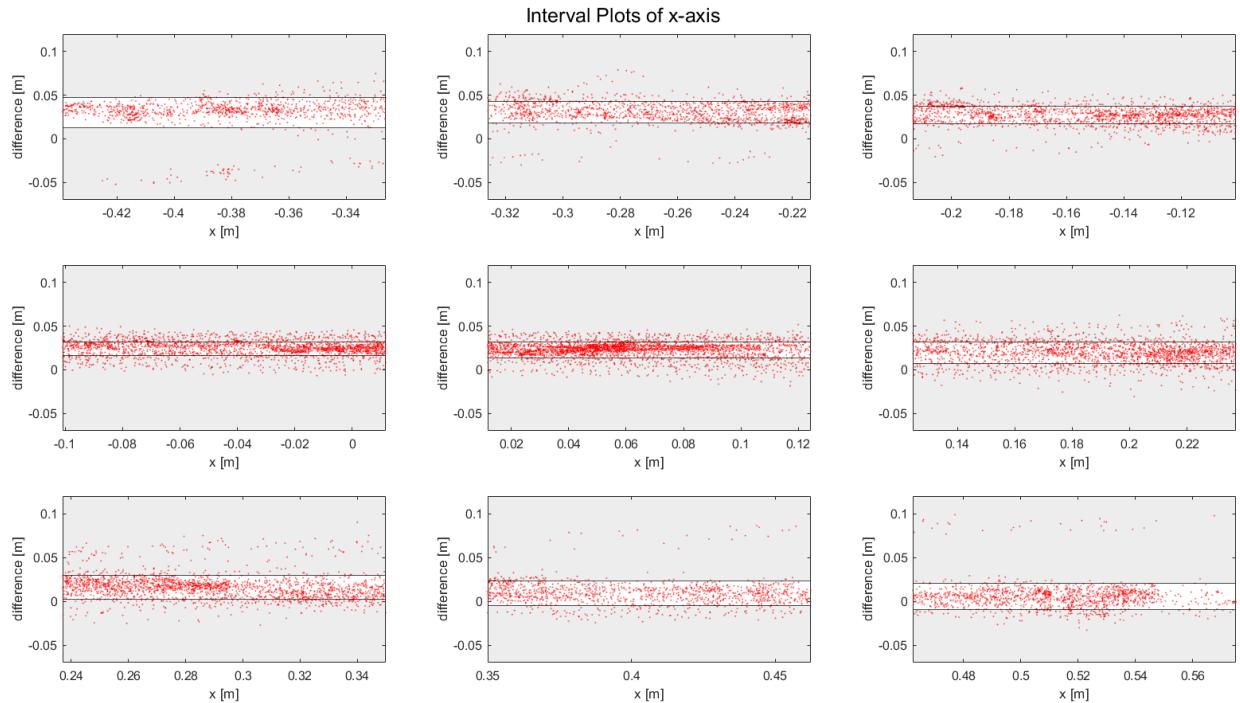


Figure 5.47: Confidence intervals plotted from distribution fitting for x direction

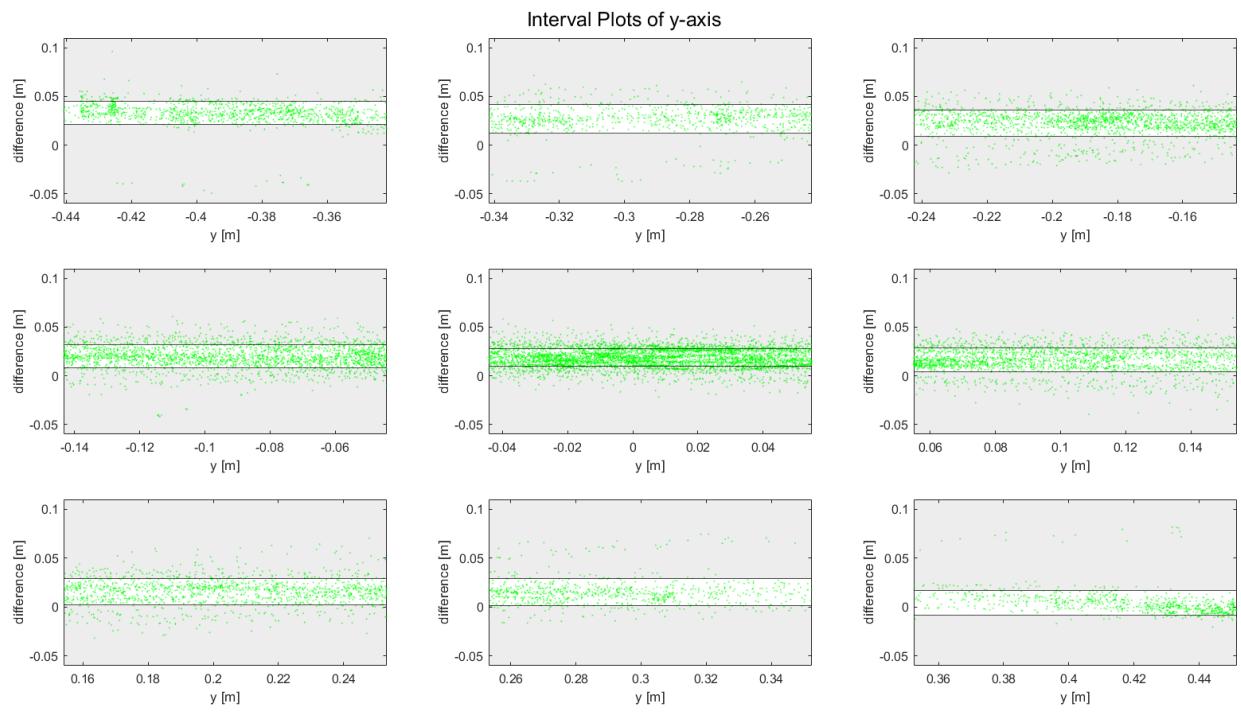


Figure 5.48: Confidence intervals plotted from distribution fitting for y direction

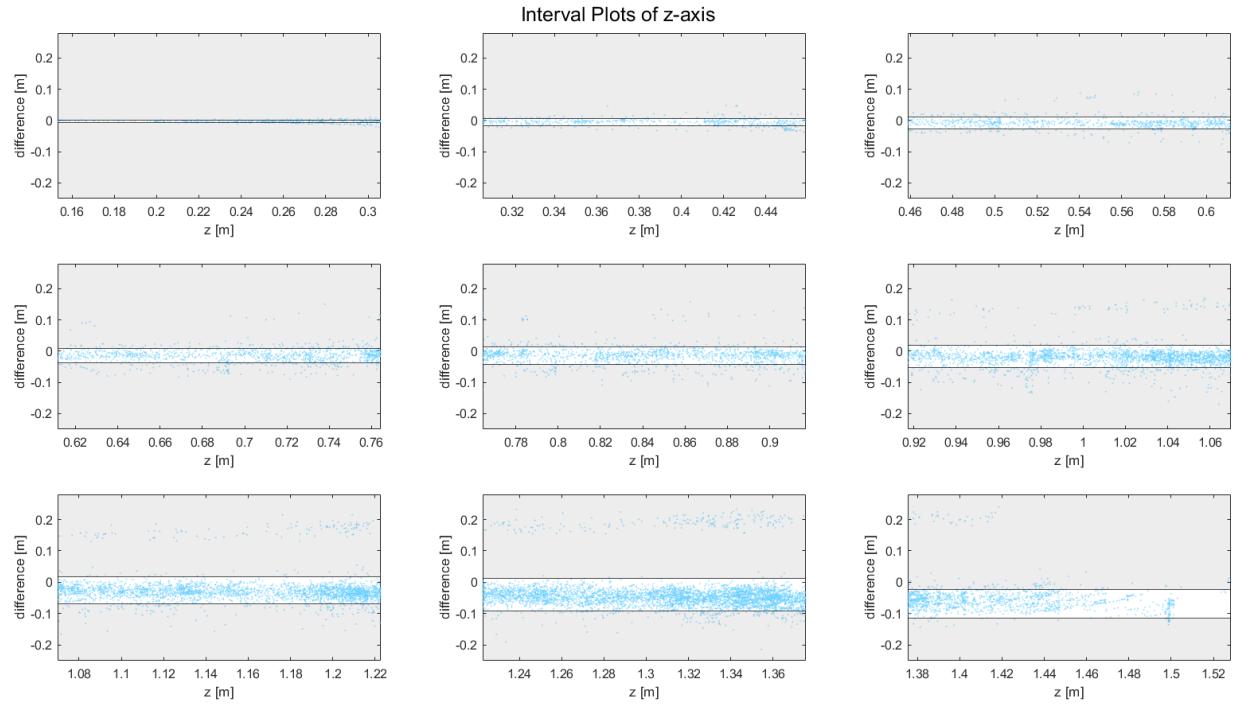


Figure 5.49: Confidence intervals plotted from distribution fitting for z direction

After this was done, the curve fitter app in MATLAB was used to parameterize the errors standard deviation as a function of QTM position. The results from the curve fitting are shown in the rest of this section, with the polynomials below the corresponding figures.

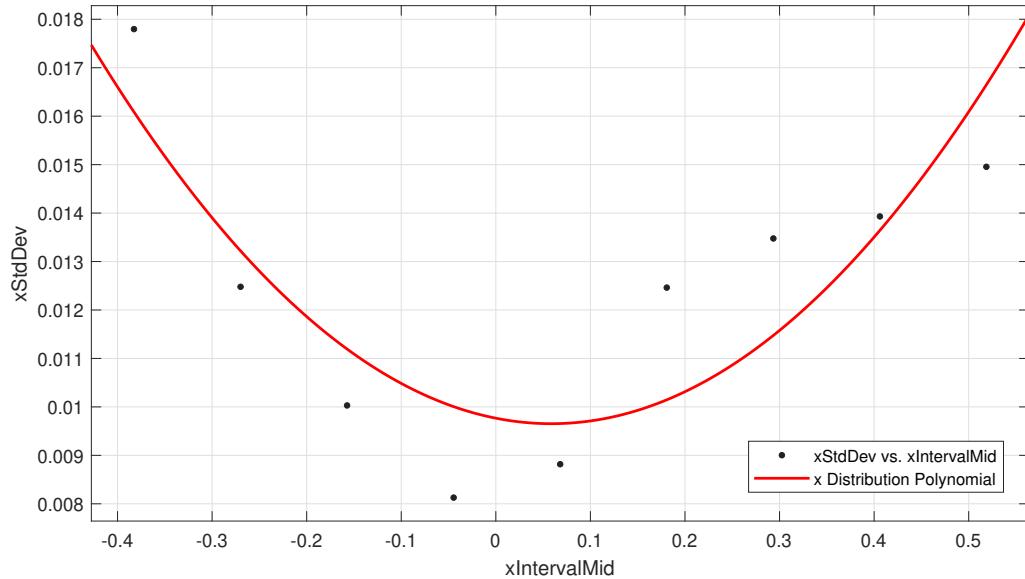


Figure 5.50: Curve fitting of standard deviation as function of distance in x-direction

$$f(x) = p_1 \cdot x^2 + p_2 \cdot x + p_3 = 0.0330 \cdot x^2 - 0.0039 \cdot x + 0.0098 \quad (5.14)$$

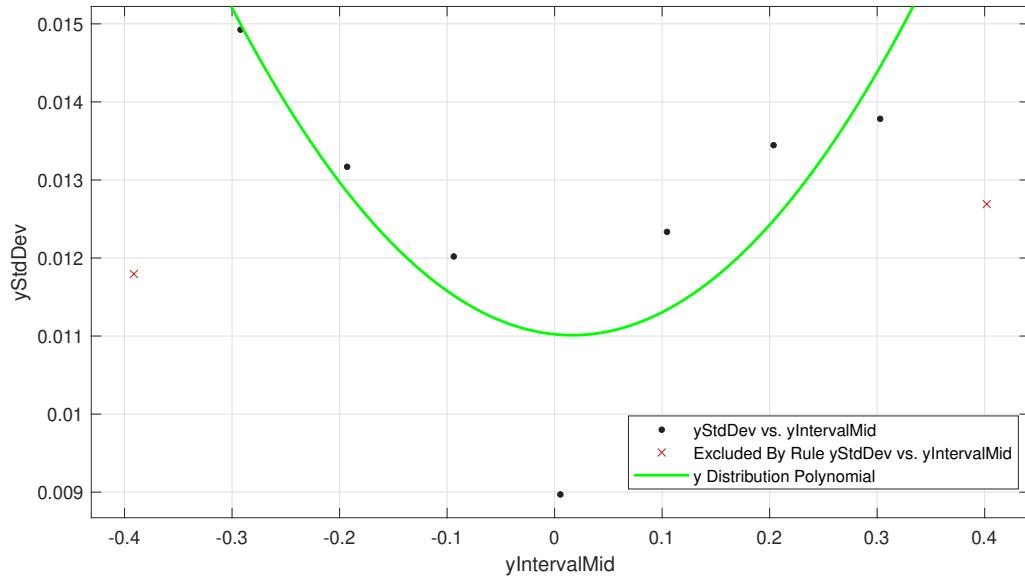


Figure 5.51: Curve fitting of standard deviation as function of distance in y-direction

$$f(y) = p_1 \cdot y^2 + p_2 \cdot y + p_3 = 0.0419 \cdot y^2 - 0.0014 \cdot y + 0.0110 \quad (5.15)$$

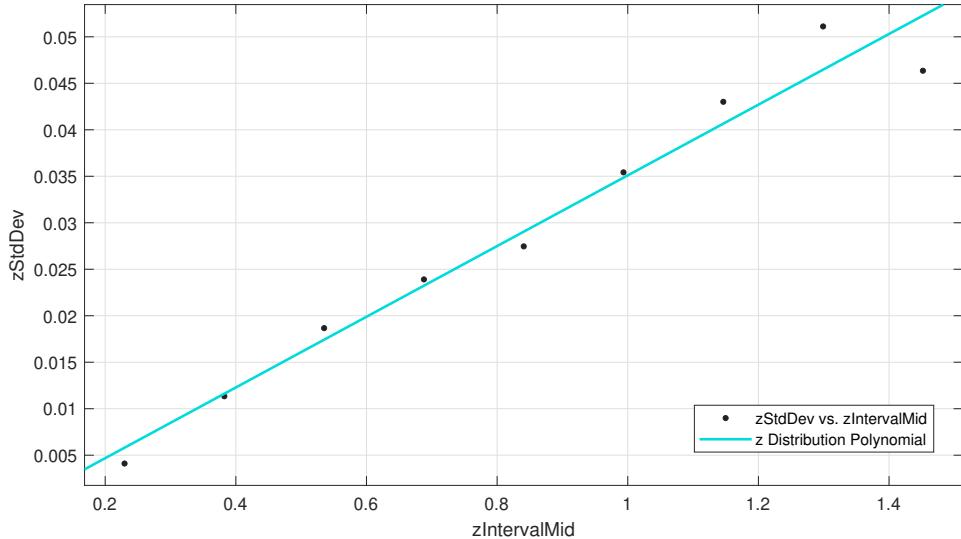


Figure 5.52: Curve fitting of standard deviation as function of distance in z-direction

$$f(z) = p_1 \cdot z + p_2 = 0.0380 \cdot z - 0.0029 \quad (5.16)$$

The goodness statistics from the curve fitter application for all three polynomials are shown in table 5.11.

Table 5.11: Goodness of fits:  $R^2$  and  $RMSE$

Statistic	x	y	z
R-square	0.7183	0.6769	0.9666
RMSE	0.0019	0.0013	0.0032

### 5.3.5 Position Kalman Filter: 1 DOF

Results in this section are regions of interest from the pose validation test #1 data, cropped from 9 to 85 seconds. The linearly interpolated data for QTM is plotted in the same figure as the original QTM data, shown in figure 5.53. The 1D Kalman filter was plotted together with the marker measurements and the QTM data, shown in figure 5.54. The confidence interval for the absolute value of the error with and without filter is shown in figure 5.55. The mean and standard deviations from the confidence intervals are shown in table 5.12

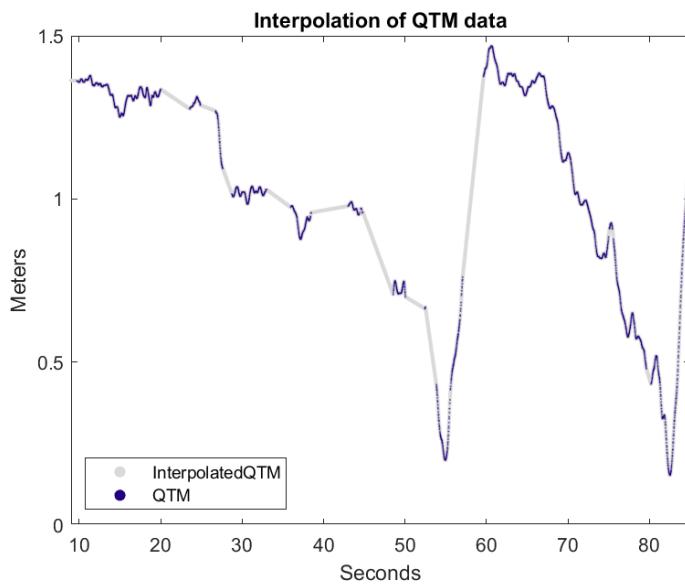


Figure 5.53: Linear interpolation of QTM data

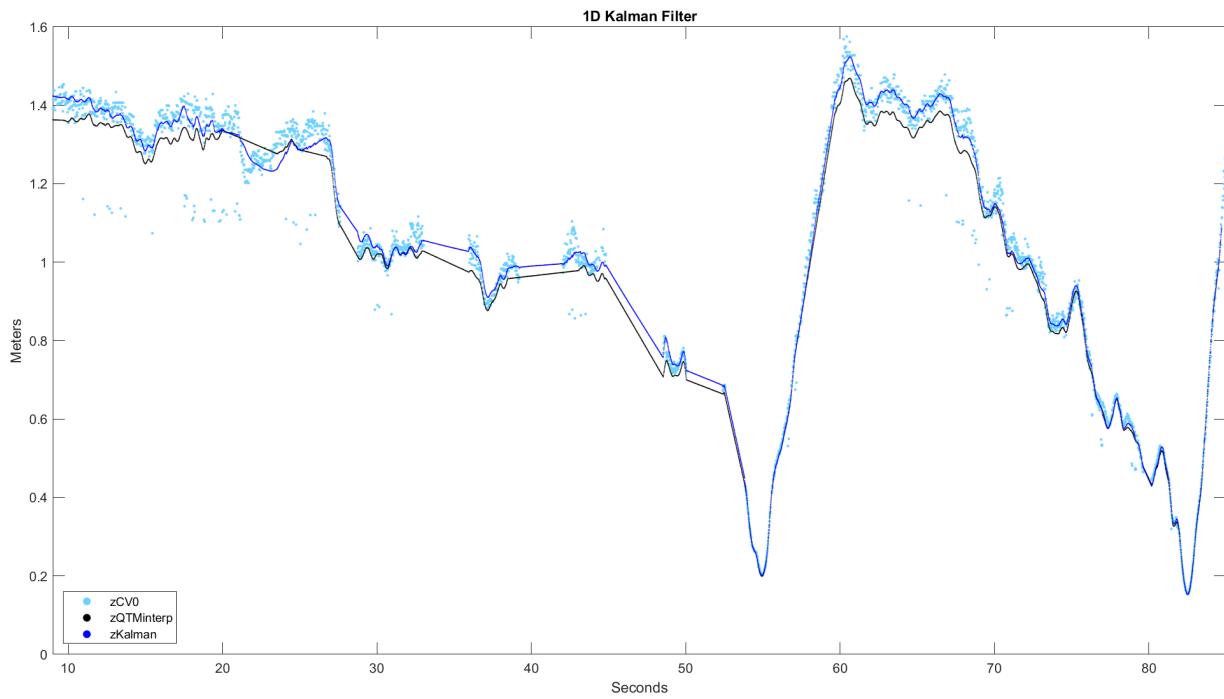


Figure 5.54: 1D Kalman filter compared to measurements and QTM reference

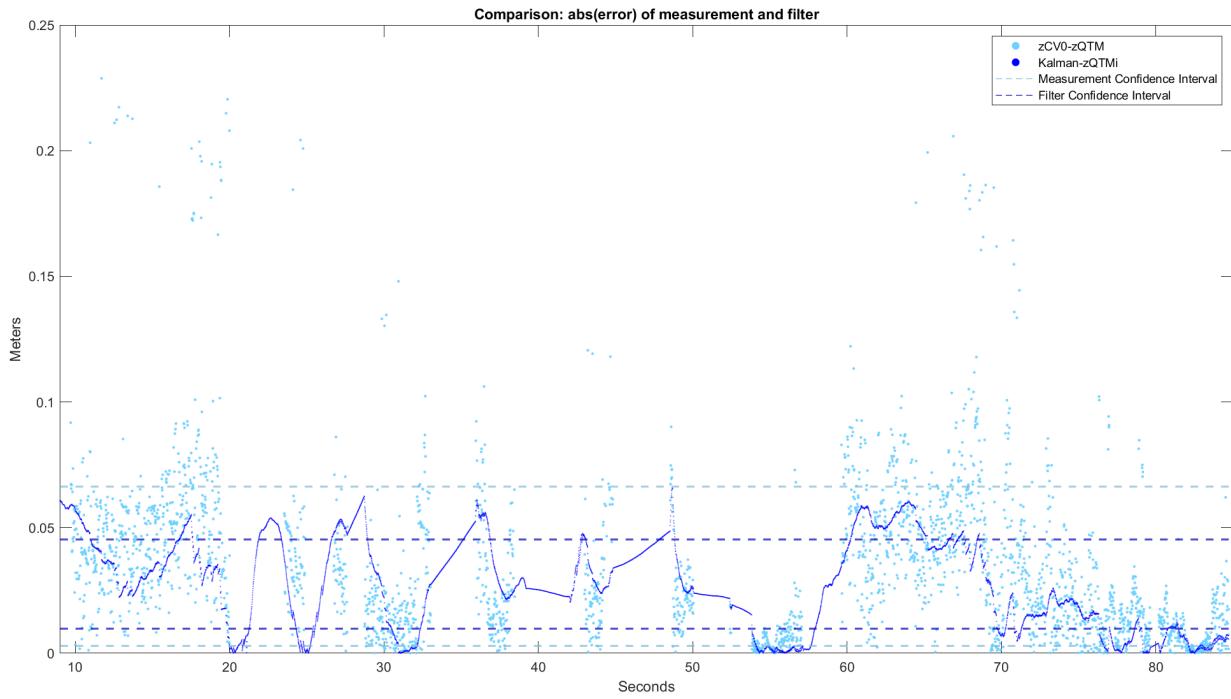


Figure 5.55: Confidence intervals for absolute value of error without and with filter

The filter was simulated for all the other tests as well, where the full interval was taken into account. These are shown in appendix E.4, with the average and standard deviation values shown in table 5.13.

Table 5.12: Average and Standard Deviation for  $|\Delta z|$  With and Without Filter: RoI from test #1

Calculation Data	Mean [mm]	Standard Deviation [mm]
z Measurement	34.603	31.608
z Estimate	27.578	17.693

Table 5.13: Average and Standard Deviation for  $|\Delta z|$  With and Without Filter: All tests

Test #	Calculation Data	Mean [mm]	Standard Deviation [mm]
Test #0	z Measurement	36.827	32.889
	z Estimate	26.705	20.812
Test #1	z Measurement	36.258	31.934
	z Estimate	28.744	18.759
Test #2	z Measurement	40.467	36.258
	z Estimate	41.825	68.440
Test #3	z Measurement	49.528	40.495
	z Estimate	37.194	29.859
Test #4	z Measurement	53.319	40.021
	z Estimate	44.222	53.411

### 5.3.6 Position Kalman Filter: 3 DOF

Results in this section are also from test #1, this time cropped from 60 to 84 seconds. For consistency, this section will also include the z-plots for the interpolation and filter as it is zoomed into a different region of interest, but all other results from the 3D filter will only be from x and y since the z-data is the same. The interpolated data for x, y and z are shown respectively in figures 5.56, 5.57 and 5.58.

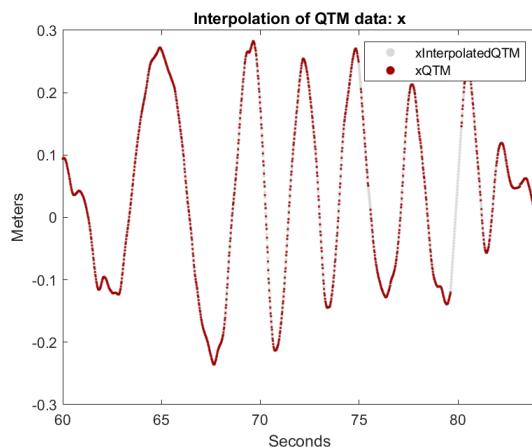


Figure 5.56: X interpolation of QTM data

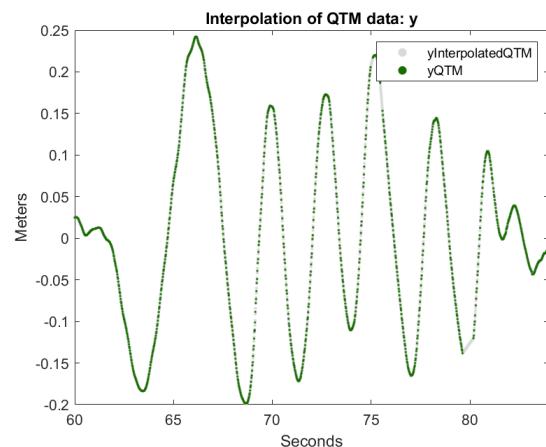


Figure 5.57: Y interpolation of QTM data

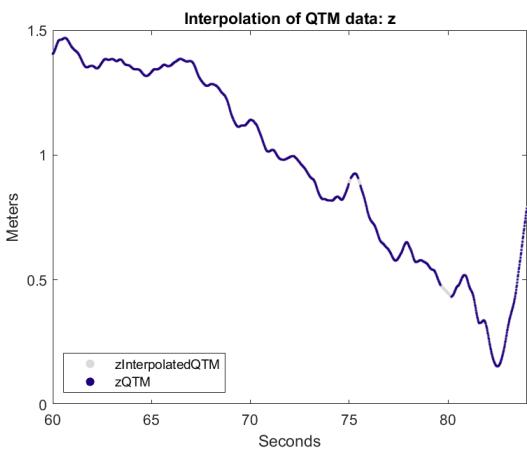


Figure 5.58: Z interpolation of QTM data

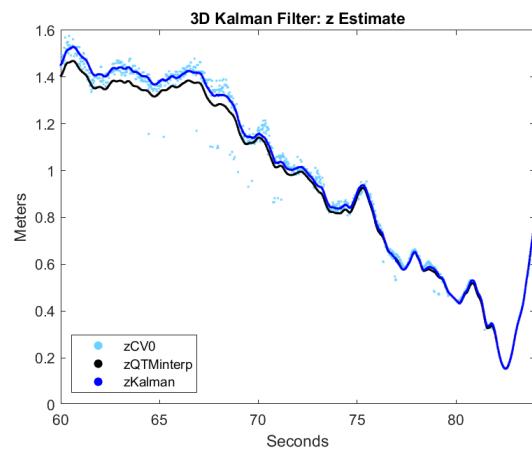


Figure 5.59: Comparison of z estimate, measurements and reference

The filter was measured like before by plotting the estimate together with the marker measurements and QTM reference, with each dimension in their own figure. The x, y and z comparison plots are shown in figures 5.60, 5.61 and 5.59 respectively.

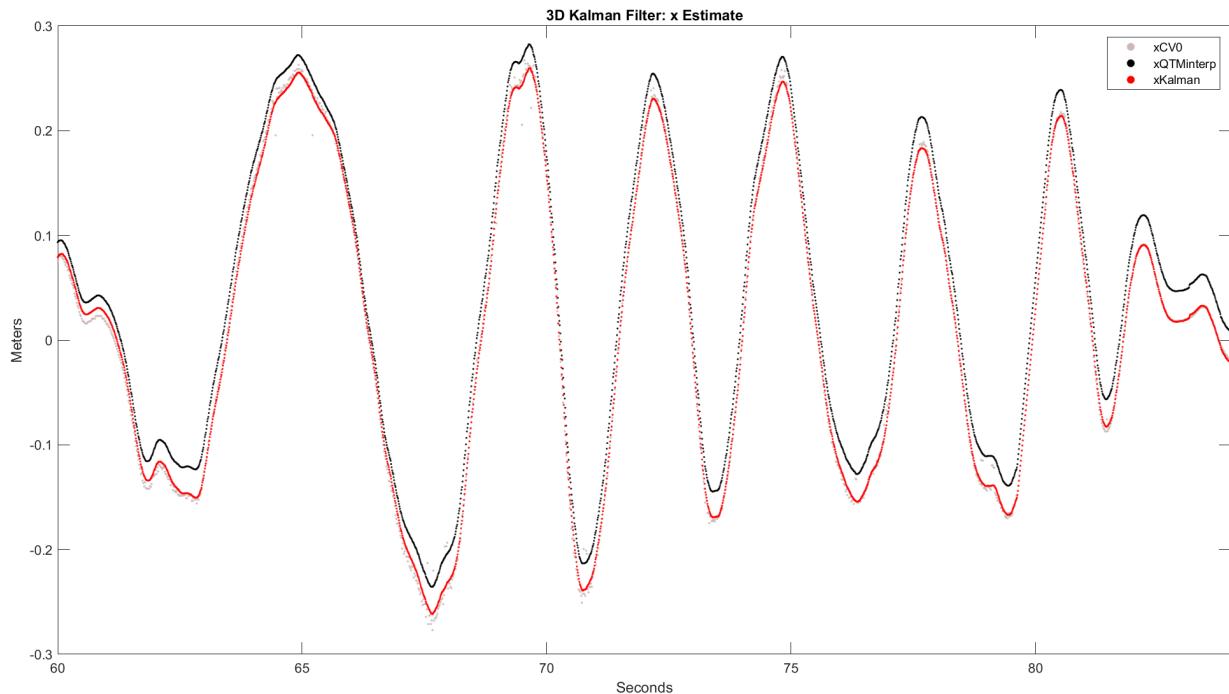


Figure 5.60: Comparison of x estimate, measurements and reference

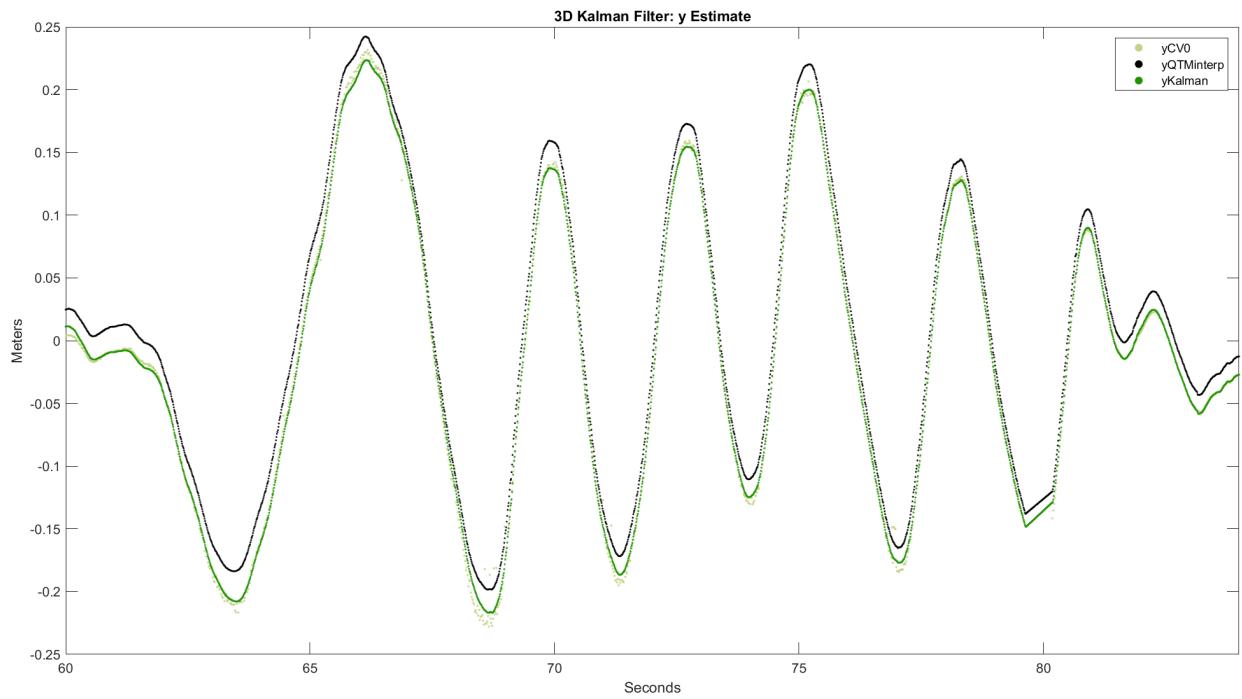


Figure 5.61: Comparison of y estimate, measurements and reference

The confidence intervals for the absolute values of the errors are also plotted like before, shown in figures 5.62 and 5.63 for x and y respectively.

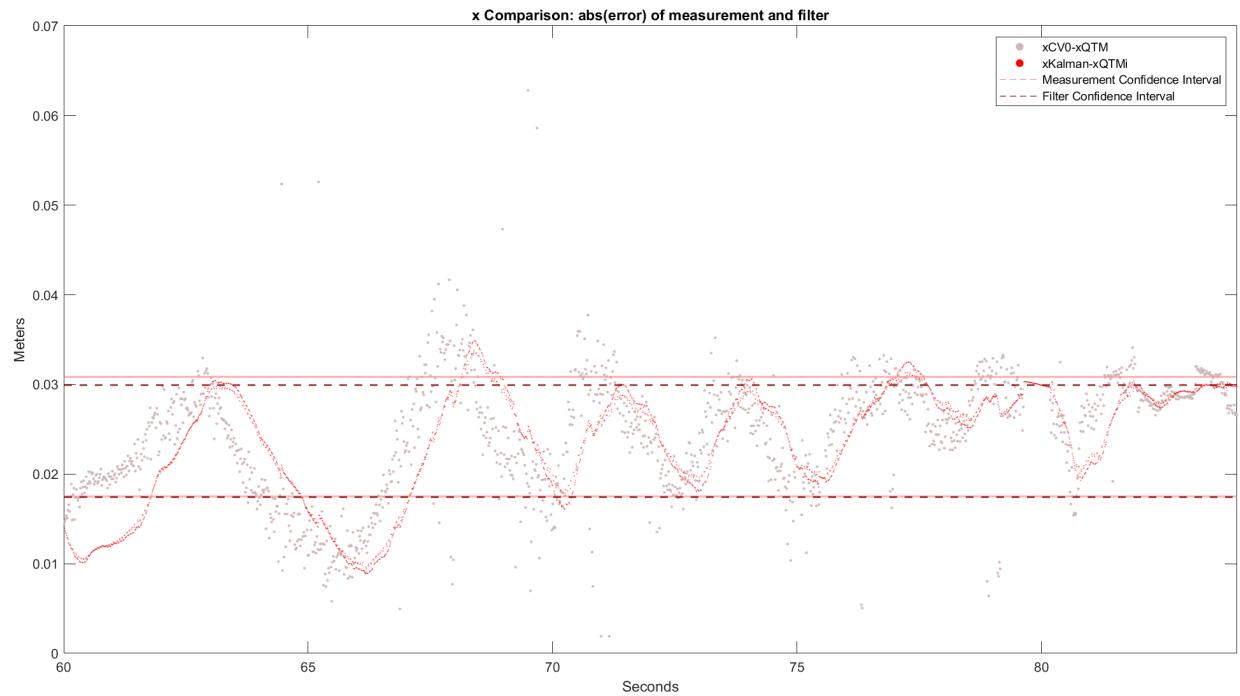


Figure 5.62: Confidence intervals for absolute value of x error without and with filter

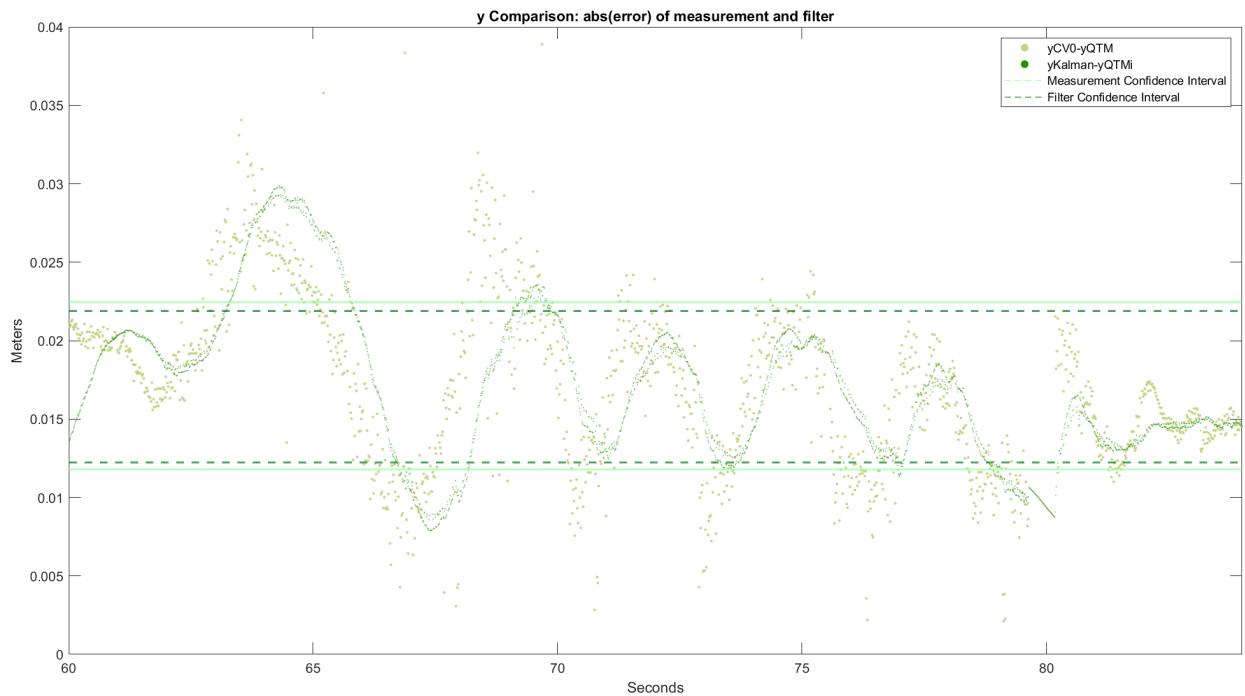


Figure 5.63: Confidence intervals for absolute value of  $y$  error without and with filter

Like before, the confidence intervals for the absolute value of the difference between the measurement/estimate and reference were plotted with the "one-sigma-range"  $\mu \pm 1\sigma$ , where the values for the region of interest from test #1 are shown in table 5.14. The same calculations were done for the entire time range for all tests, where the numeric results are shown in table 5.15. The results for the rest of the tests over the full time range are shown in appendix E.5.

Table 5.14: Mean and Standard Deviation for  $|\Delta x|$  and  $|\Delta y|$  With and Without Filter: test #1 RoI

Calculation Data	Mean [mm]	Standard Deviation [mm]
x Measurement	24.174	6.6923
x Estimate	23.685	6.2373
y Measurement	17.126	5.3323
y Estimate	17.062	4.8443

Table 5.15: Average and Standard Deviation for  $|\Delta x|$  and  $|\Delta y|$  With and Without Filter: All Tests

Test #	Calculation Data	Mean [mm]	Standard Deviation [mm]
Test #0	x Measurement	21.814	13.445
	x Estimate	24.595	22.936
	y Measurement	22.705	12.177
	y Estimate	31.623	27.270
Test #1	x Measurement	23.916	11.154
	x Estimate	29.082	20.332
	y Measurement	18.131	9.692
	y Estimate	39.093	49.379
Test #2	x Measurement	21.564	10.765
	x Estimate	45.605	66.903
	y Measurement	19.080	9.855
	y Estimate	49.985	107.15
Test #3	x Measurement	28.721	17.167
	x Estimate	34.226	32.594
	y Measurement	25.462	20.794
	y Estimate	31.258	33.679
Test #4	x Measurement	22.025	11.968
	x Estimate	21.041	10.398
	y Measurement	23.103	11.493
	y Estimate	21.645	10.970

## 5.4 Server

### 5.4.1 Message transfer test

Using the raspberry pi 4B connected to Ethernet with the BMI088 shuttleboard, a data-stream of IMU readings were sent to the server program running on a separate computer to be transferred and read from a python script which also sent simulated motor values. The purpose of this test was to simulate a real running cycle of the arena, where the drone would send messages of readings from the IMU, and receive messages to run the motors from the AI program. During the test, messages from the IMU could be shown on the python program, and messages for the motors could also be read from the raspberry pi 4B, these messages were transferred between the server on the sending and receiving interval of 10 [ms] without any loss of messages.

### 5.4.2 Delay test

From a python client, the delay between when a message arrived compared to when a message was expected to arrive was taken, the messages were sent at an interval of 10 [ms] which was an IMU reading from a raspberry PI connected with Ethernet, the message is then sent to the server and

passed on to the python client. From this test, it was found that there was an average of 0.361 [ms] delay, and a median of 0.357 [ms]. The shortest measured delay was 0.268 [ms] and the longest being 0.473[ms]. However the first message arrived with a delay of 10.7[ms]. During the test, 459 messages were recorded, with no loss of messages except for the first as mentioned.

## 5.5 Actuator

### 5.5.1 USB communication

The test code using the LUFA library for the microcontroller functioned well. Data was received as a 16-bit byte from VScode, which then turned on or off the LED and sent the same message back. The resulting code size was 4kB in flash memory, where the ATMEGA32u4 has 32kB of flash memory [9].

### 5.5.2 Endstop Button

The endstop button was checked for its functionality by polling, a simple python script was written for this. When the button was depressed, the program detected the press which meant that the button was functional and working as intended. The code for this test can be found in G.6.3

### 5.5.3 Encoder

The functionality of the newly implemented encoder was tested. It was found that when the arena was not screwed down then shaking the arena without rotating it had a massive affect on the reading, this can be seen in the graph 5.64. However, when the Arena was mounted, shaking it had no affect on the reading.

The encoder was then read from the RAW\_ANGLE registers during a rotation of the motor, this can be seen in the graph 5.65 as the azimuth motor runs at a PWM value of 30 for 4 seconds first in the counter-clockwise direction, then stops for 1 second before repeating the same movement in the opposite direction. The jump is due to the shift in position of the magnet, similarly to the arena being shaken in the graph 5.64. Physically, it could be seen that the motor was shaking the actuator system which holds the vertical actuator.

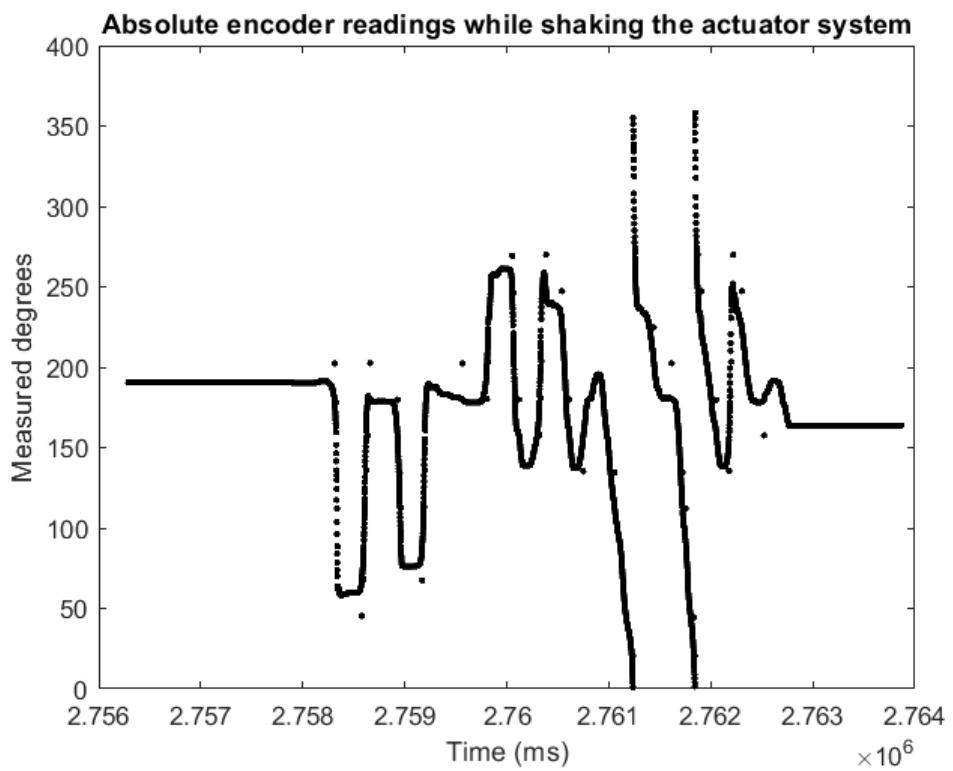


Figure 5.64: Graph showing readings from the absolute encoder during the arena actuator being shaken while not screwed down

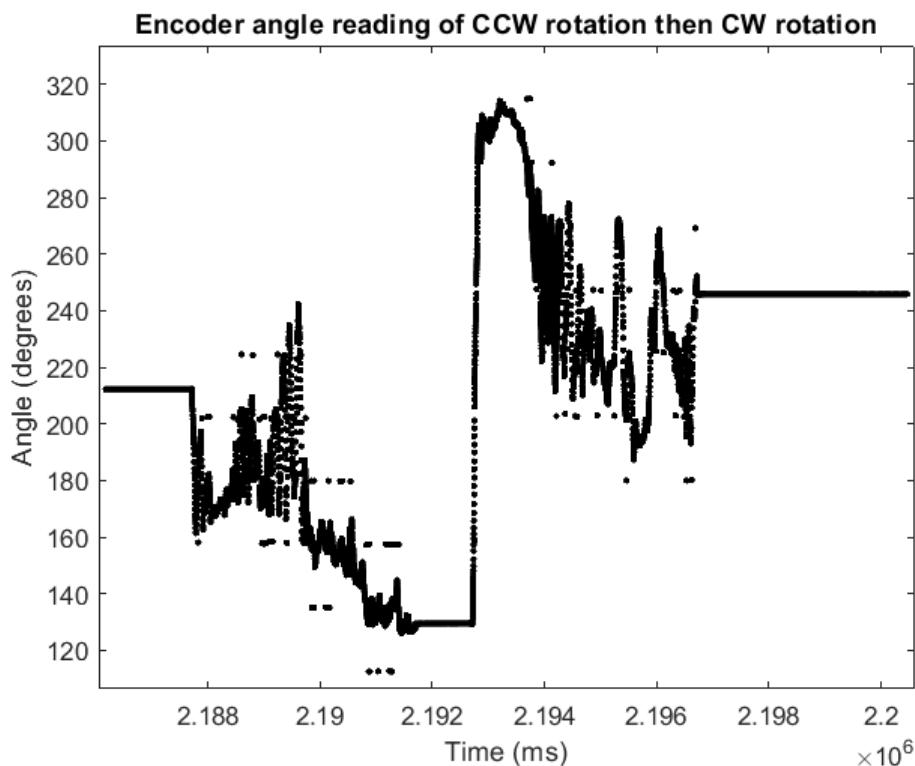


Figure 5.65: Graph showing readings from the absolute encoder while the azimuth motor runs with a PWM value of 30 for 4 seconds in the counter-clockwise direction, stopping for 1 second, then repeating the same movement in the opposite direction

## 5.6 Drone Hardware and Software

### 5.6.1 IMU Readings

These readings were done using the code in Appendix G.7.2.

#### 5.6.1.1 Accelerometer gravity readings

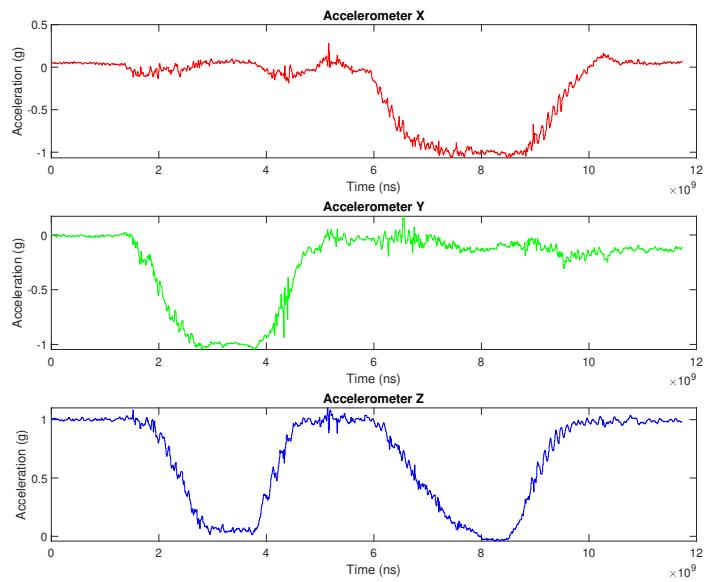


Figure 5.66: Graph showing readings from the accelerometer when the IMU starts flat, then rotate to Y towards earth, then rotated to X towards earth

#### 5.6.1.2 Gyroscope movement readings

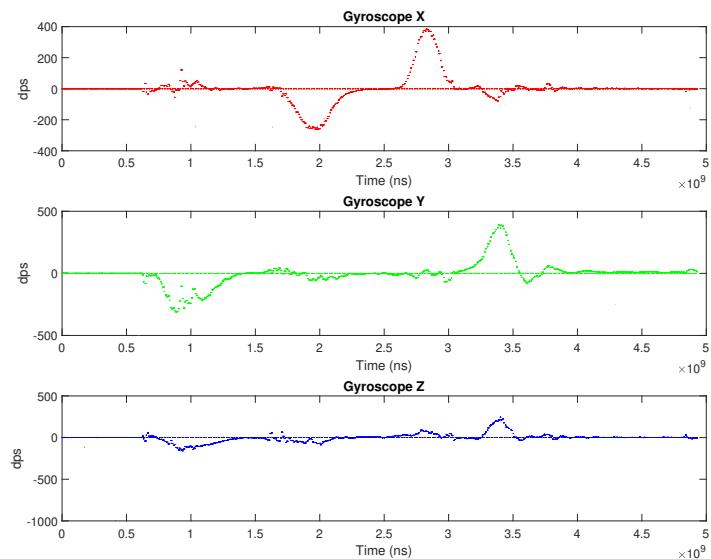


Figure 5.67: Graph showing readings from the gyrometer during rotations individually in all axis

## 5.6.2 Buildroot

On boot, the CM4 successfully connects to WiFi. When running BMItest, the gyroscope and accelerometer IDs are read with expected values.

```
jan@jan-System-Product-Name:~$ ssh root@128.39.202.142
root@128.39.202.142's password:
# killall pigpiod
# cd /
# cd usr
# cd bin
# ./BMItest
spiHandle:0
bytes sent: 01111101 00000100
bytes read: 11111101 11111111
ACC_PWR_CTRL: 00000000
Gyro ID: 00000000 00001111
Accel ID: 00011110 00100011
```

Figure 5.68: Successful BMI connection

Listing 5.1: CM4 resource use

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/root	109525	99773	1151	99%	/
devtmpfs	899008	0	899008	0%	/dev
tmpfs	933920	0	933920	0%	/dev/shm
tmpfs	933920	72	933848	0%	/tmp
tmpfs	933920	44	933876	0%	/run

	total	used	free	shared	buff/cache	...
	available					
Mem:	1824	41	1737	0	47	...
1756						
Swap:	0	0	0			

## 5.7 Mechanical Drone Design

### 5.7.1 Geometric Design

As mentioned in section 4.11.2, the drone was desired to slide correctly into the actuator hole in the bottom of the arena without pivoting about the bottom contact point shown in figure 4.30. The designs were made for fitting the hole, and the latter ability was checked mathematically through rotation of the CoG coordinates from the drone oriented reference frame placed at the pivot point. In order to do this, the angle formed by the drone's tilt about the pivot point was found through trigonometry, utilizing the x and y distances from the pivot point from the line tangential to the contour of the bottom:

$$\theta_1 = \tan^{-1} \left( \frac{dY}{dX} \right) = \tan^{-1} \left( \frac{21.335969}{99.794854} \right) = 12.06805^\circ \quad (5.17)$$

Adding this to the  $12.76^\circ$  metal plate angle  $\theta_2$  [14] gave the total angle of  $\theta \simeq 24.8281^\circ$ .

Using this yielded the following results in [mm] for the remaining calculations:

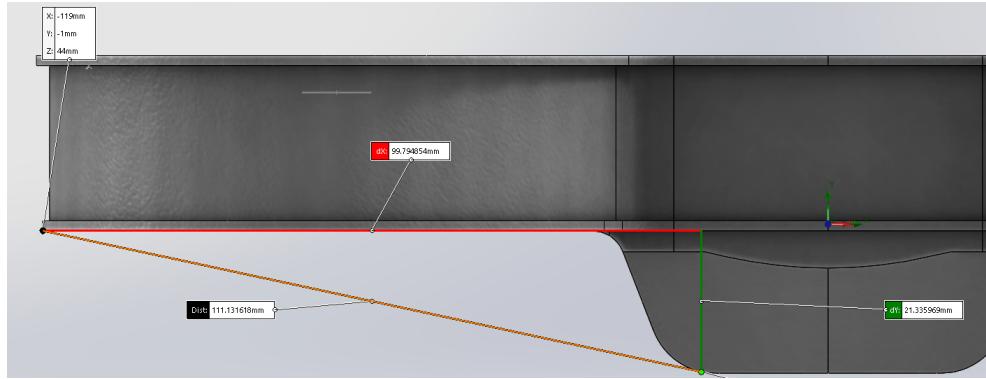


Figure 5.69: Distances from tangent line intersection with outer contour to propeller guard edge

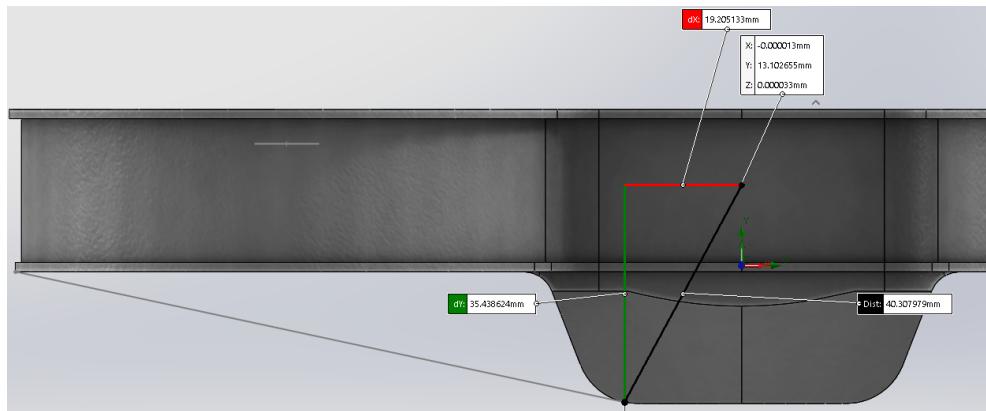


Figure 5.70: Distances from tangent line intersection with outer contour to CoG

$$\begin{aligned}
 \begin{bmatrix} x'_d \\ y'_d \end{bmatrix} &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} dX \\ dY \end{bmatrix} \\
 &= \begin{bmatrix} \cos 24.8281 & -\sin 24.8281 \\ \sin 24.8281 & \cos 24.8281 \end{bmatrix} \begin{bmatrix} 19.2051 \\ 35.4386 \end{bmatrix} = \begin{bmatrix} 2.5495 \\ 40.2272 \end{bmatrix}
 \end{aligned}$$

Technical drawings were made for the final mechanical designs, shown in appendix A.3, where the assembly's total weight except for the PCB is around 207 grams.

### 5.7.2 3D Printing

The produced tent for the 3D-printer is shown in figure 5.71.

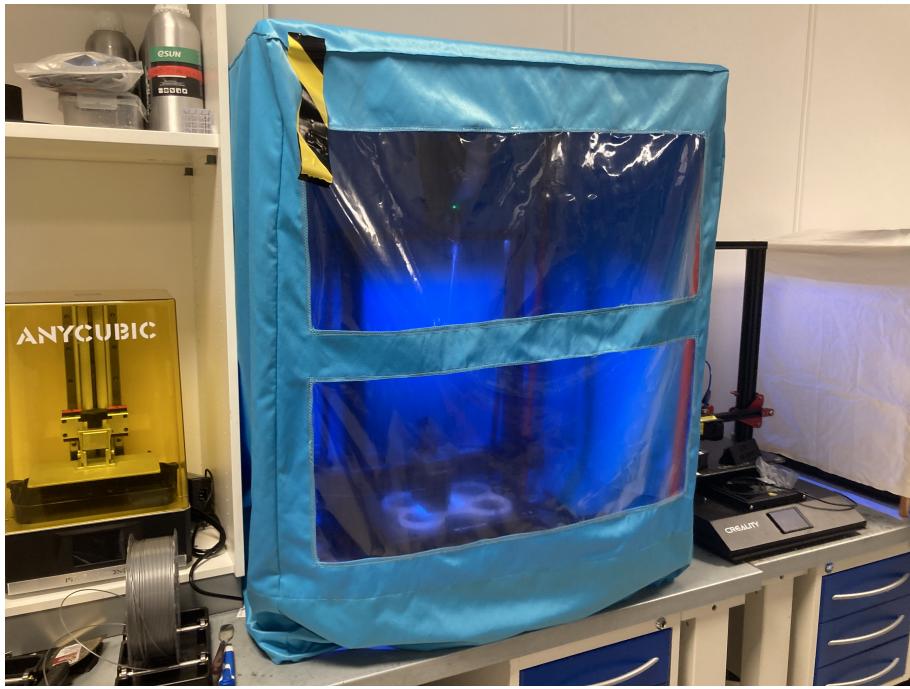


Figure 5.71: Beaver Nylon tent with transparent oilcloth window and support structure

Initially, the polypropylene print results were reasons for dismay. The first iterations suffered from poor bed adhesion, warping, and delamination. In order to obtain successful prints, the print configuration had to deviate from the manufacturers recommended settings [94]. To combat delamination, the nozzle temperature was increased from  $220^{\circ}$  to  $230^{\circ}$ . The recommended settings presupposes that a PPprint P-Surface 141 is used, which experiences stronger adhesion after cooling down the first print layer. A P-Surface 141 was ordered, but arrived too late in the process to be of any use. The group was able to finish printing by experimenting with packing tape surfaces. Initially a Biltema packing tape was used, but better results were obtained when switching to a premium packing tape by Tesa (model 4280). The comparatively expensive tape was perceived to have better adhesion to the bed at higher temperatures, justifying the 10x more expensive price point. The key to making the packing tape surface work was balancing the temperature of the bed. Higher temperatures upwards of  $90^{\circ}$  gave excellent adhesion to the tape, but also reduced the strength of the tape glue, resulting in the tape being torn off the print bed by warping material. The additional adhesion of filament to tape had little utility if the tape itself did not stick to the tray. A balance was found with a bed temperature of  $70^{\circ}$  as recommended by PPprint [94], but instead of cooling down after the first layer,  $70^{\circ}$  was sustained throughout the entire print. It was noticed that parts of the print would adhere better to certain parts of the print tray. An investigation was launched and the print tray was found to not always be completely uniform in temperature. The infrared thermometer showed that for some print sessions temperatures could deviate by as much as  $\pm 5^{\circ}$  with the front of the tray reading hotter values, and the rear reading colder. It is worth noting that this problem was not always present, as readings done after finishing the final print were more uniform across the surface. It was also noticed that multiple prints also had problems with warping towards the rear of the printer, while the front facing part of the print was in excellent condition. These two problems were addressed by moving the the print location slightly more towards the front, and ensuring the enclosure cloth touching the ground all the way around the printer.

The small test prints made during empirical testing are shown in figure 5.72, illustrating the increased print quality with the crafted tent and settings from empirical testing, when compared to the recommended settings [94].



Figure 5.72: Settings used for final drone print (left) and recommended settings without enclosure (right) [94]



Figure 5.73: Rearmost corners lifting the packing tape off the print tray due to warping

The resulting printer settings used for the Artillery slicer is shown in table 5.16.

Table 5.16: Artillery Slicer 3.0.3 settings for Artillery X4 Plus printer using P-Filament 721

Setting	Value
Mode	Expert
Filament setting config startpoint	Artillery TPU @ X4
First layer height	0.3 [mm]
Fill density	5%
Brim type	Outer and inner
Brim width	25 [mm]
Generate support material	Enabled
First layer expansion	20 [mm]
Raft layers	1 layer
Raft expansion	0 [mm]
Top contact Z distance	0.25 [mm]
Print speed	30 [ $\frac{mm}{s}$ ]
First layer speed	25 [ $\frac{mm}{s}$ ]
Speed of object first layer over raft interface	25 [ $\frac{mm}{s}$ ]
Nozzle temperature	230° for all layers
Bed temperature	70° for all layers
Disable fan for the first	0 layers
Filament type	PP
Soluble material	Enabled

With this setup, the final print quality is shown in figures 5.74 and 5.75.



Figure 5.74: Hull bottom print result

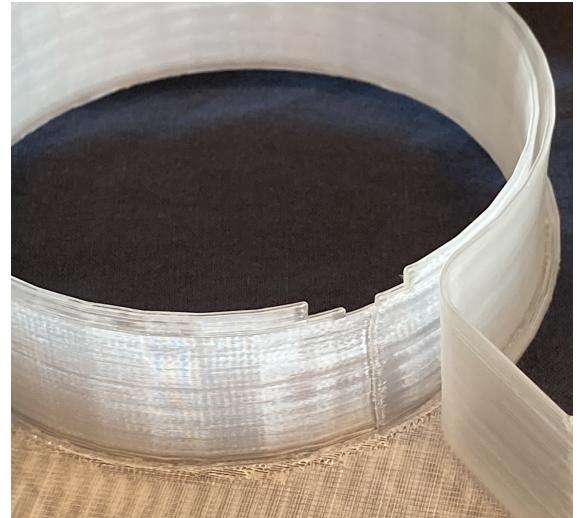


Figure 5.75: Hull top detail print result

### 5.7.3 Computational Fluid Dynamics Simulations

The force from the simulations in Y direction were simulated for both this project's drone design and the previous design and plotted as function of the propeller speed, as shown in figures 5.86 and 5.87.

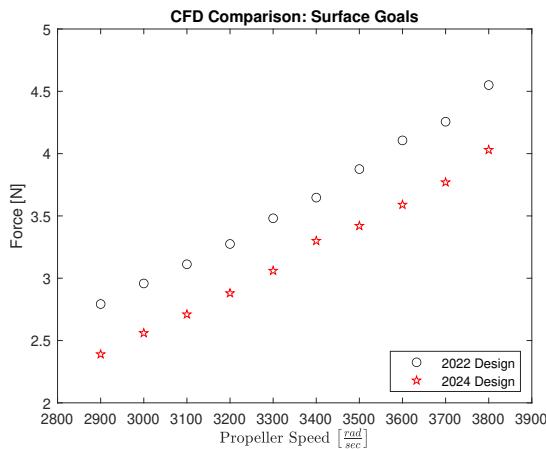


Figure 5.76: CFD Comparison: Surface Goals

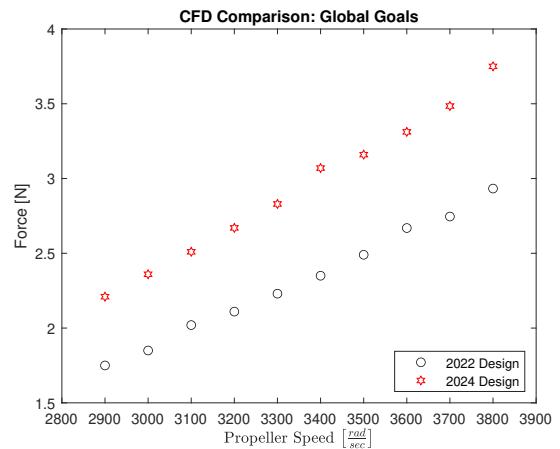


Figure 5.77: CFD Comparison: Global Goals

The high pressure zone shown in 5.78 plays a big part in the propeller thrust figure of the older drone design. However, since the air has nowhere to go, the air pushes back against the drone hull, leading to parasitic drag. Using the relationship between force and the pressure difference from figure 5.78, the force lost to the propeller arm for the 2022 design is found:

$$F_{2022} = \Delta P \cdot A = (10727.3[Pa] - 101325[Pa]) \cdot 0.0007[m^2] = 0.2814[N] \quad (5.18)$$

As this happens for all four arms, the total force of thrust lost is  $4 \cdot F_{2022} = 1.1256[N]$  at  $3300 \frac{rad}{s}$  as shown in figure 5.78.

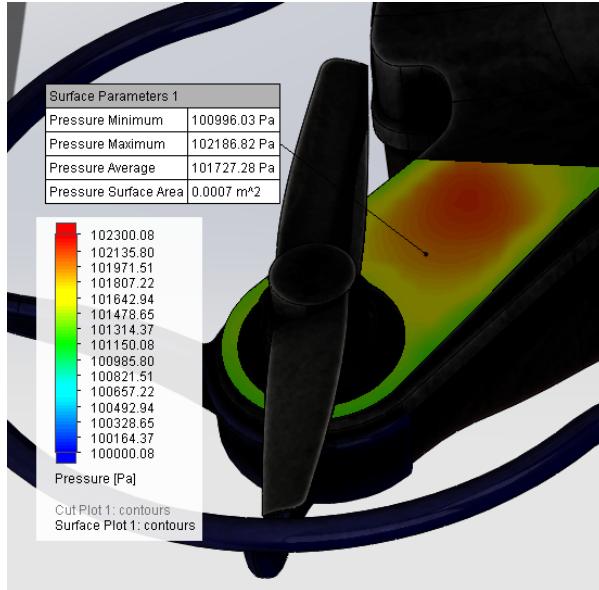


Figure 5.78: 2022 surface pressure at  $3300 \frac{\text{rad}}{\text{s}}$

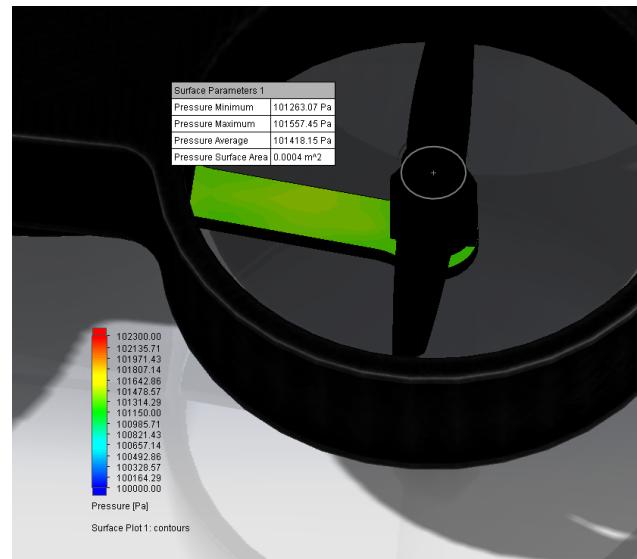


Figure 5.79: 2024 surface pressure at  $3300 \frac{\text{rad}}{\text{s}}$

Likewise, the force lost due to the frame on this project's design was found with the same relationship with the pressures from figure 5.79:

$$F_{2024} = (101418[\text{Pa}] - 101325[\text{Pa}]) \cdot 0.0004[\text{m}^2] = 0.0372[\text{N}] \quad (5.19)$$

with the total lost force  $4 \cdot F_{2024} = 0.1488[\text{N}]$ .

The flow trajectories from the simulations were also extracted from the simulations, where a view of the downwash from below the drones can be seen from both drones in figures 5.80 and 5.81.

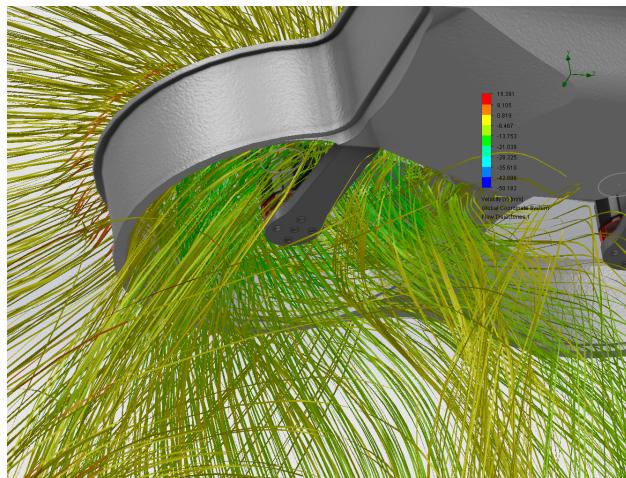


Figure 5.80: Flow trajectory 2024 design

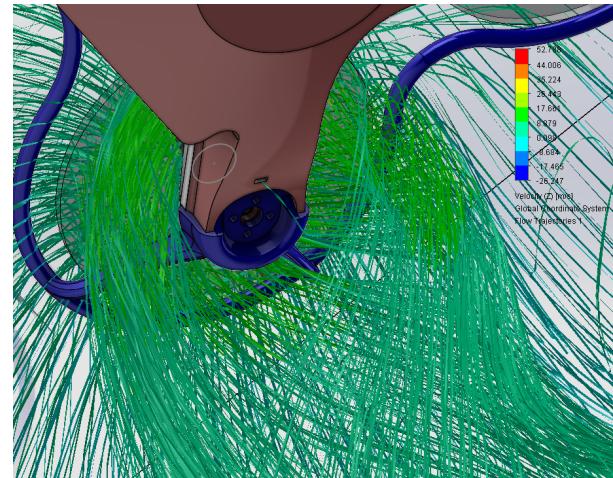


Figure 5.81: Flow trajectory 2022 design

#### 5.7.4 Impact test simulations

The figures in this section are shown with true deformation scale and with stress plots on the surfaces.

##### 5.7.4.1 "Flat Fall": 1.5 meter fall landing square onto arena floor

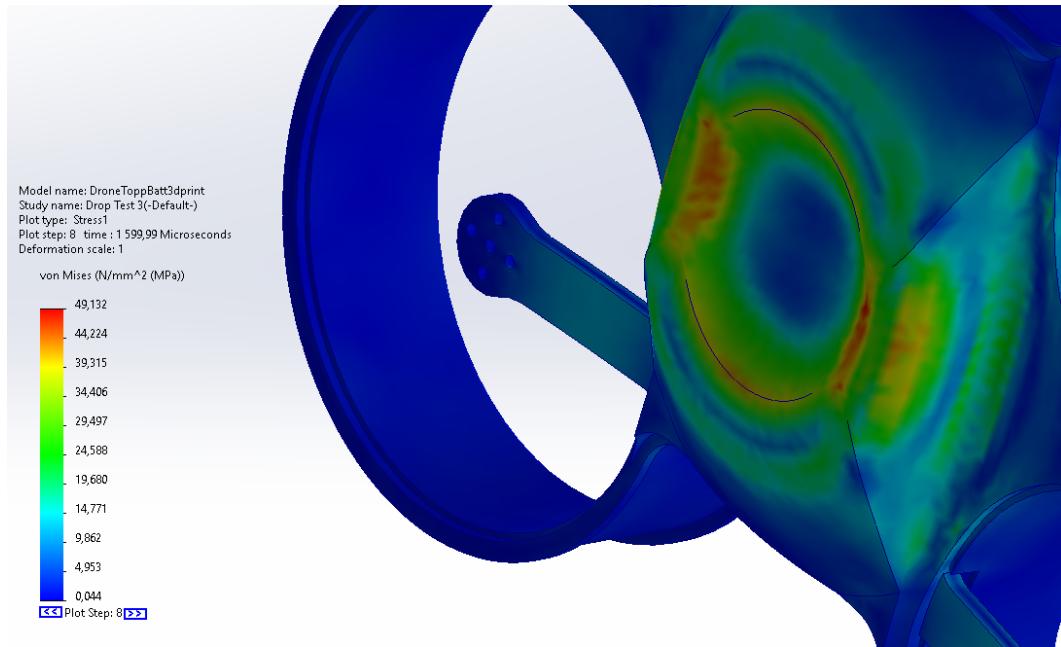


Figure 5.82: Peak stress and deformation at 1600 microseconds, rotated for bottom view

##### 5.7.4.2 "Vertical Fall": 1.5 meter fall with 90° rotation - 1 propeller guard impact

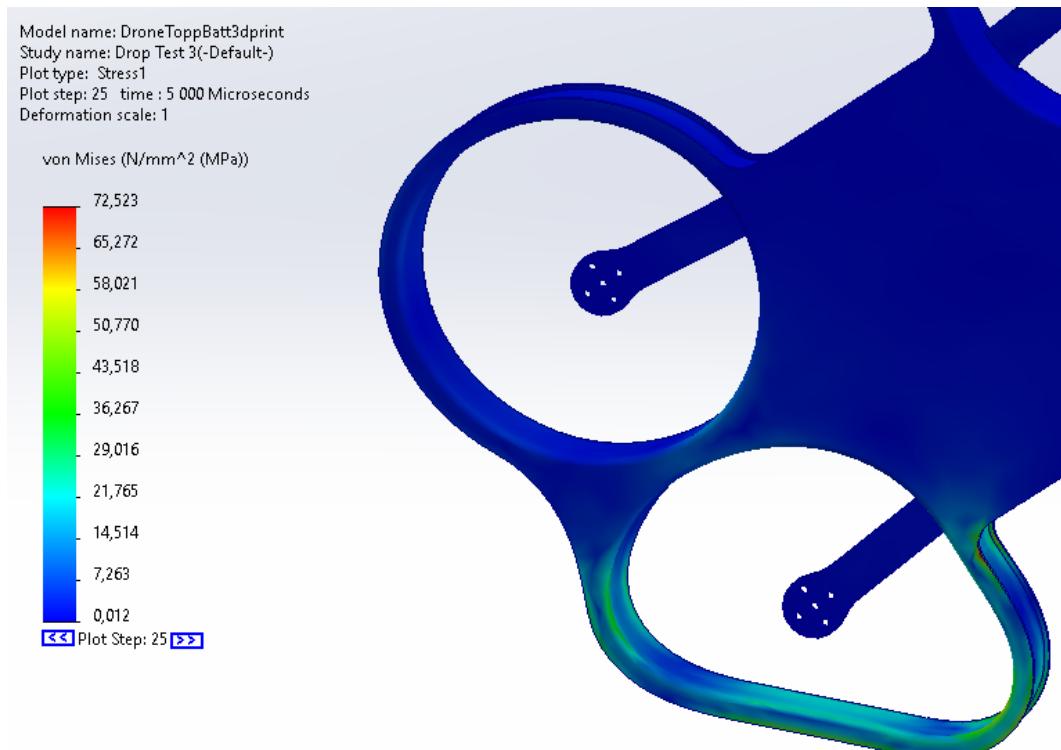


Figure 5.83: Peak stress and deformation at 5000 microseconds

#### 5.7.4.3 "Tilted Fall 1": 1.5 meter fall with 45° rotation - 1 propeller guard impact

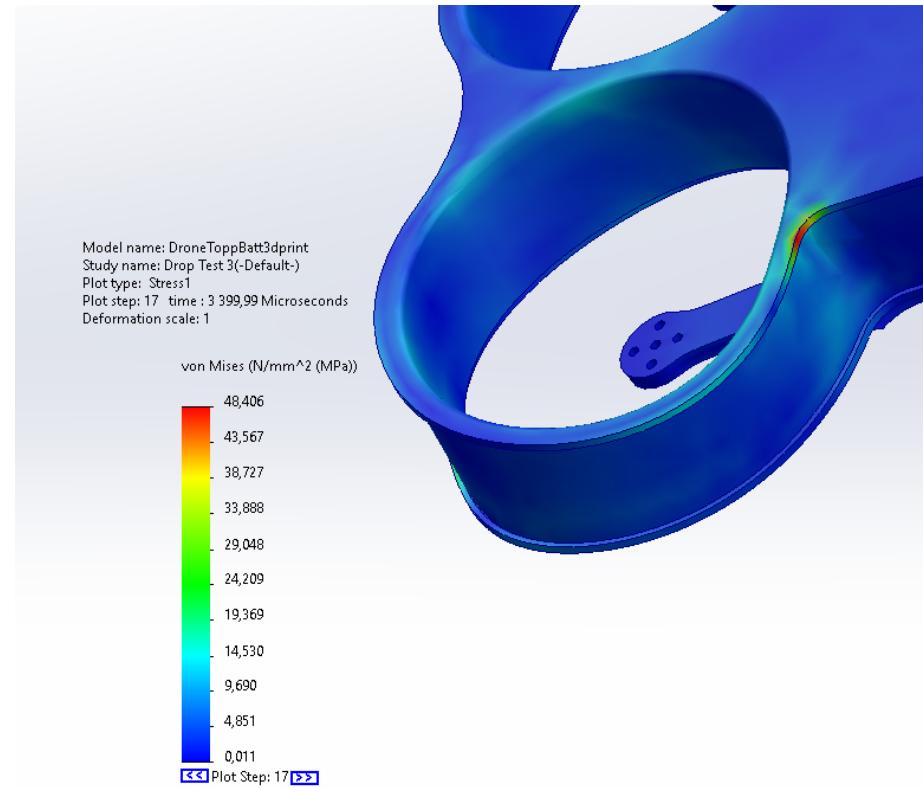


Figure 5.84: Peak stress and deformation at 4000 microseconds

#### 5.7.4.4 "Tilted Fall 2": 1.5 meter fall with 45° rotation - 2 propeller guards impact

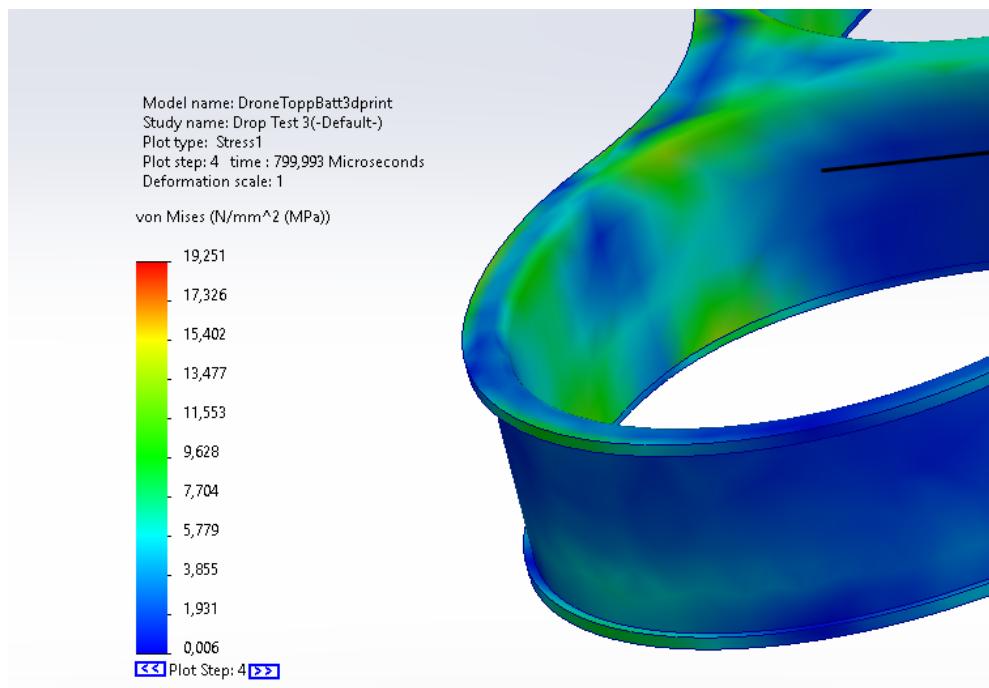


Figure 5.85: Peak stress and deformation at 800 microseconds

#### 5.7.4.5 Verifying stress concentration

Figures 5.84 and 5.82 both exhibit the same stress concentration at the point where the prop guard merges with the front of the drone. A suspicion arose that this was an erroneous FEM analysis point due to the simplified meshing used in the drop tests. A static stress analysis was initiated, and the mesh was reproduced. A 100 [N] force was applied under the propguard to simulate impact, while the opposite half of the drone was considered fixed geometry. The static simulation was repeated with 12, 8, 5, and 3[mm] mesh sizes to see if the stress converged or diverged. For 5 and 3[mm] the stress converged towards 60 [MPa], confirming that the drop test stress value is accurate.

#### 5.7.5 Physical Tests

The design was self righting when tested inside the arena and slid down the arena floor, but the variable friction of the metal plates means lubrication is required to reach the charging station reliably.

During the arena drop tests, no damage was sustained. The 5[m] drop test inflicted permanent damage.



Figure 5.86: Hull surface crack after 5[m] drop test



Figure 5.87: Bottom lid after 5[m] drop test

While damaged, the drone was still in flyable shape even after three 5 meter drop tests. The 3[mm] thick PLA frame was specifically not glued to allow the hull to deform and rotate about the arm support point.

# 6 Discussion

## 6.1 System Architecture and Design

A state estimator is desired for the drone RL arena, due to its numerous benefits. Central are the fact that it estimates the hidden state of the drone, providing more information than sensors alone, along with the fact that it can update the estimate with a constant and higher rate than the sensors. The machine vision system was chosen as the low rate measurement for the estimator, given the fitting properties. Though it may have some noise in the measurements, it is constant and will not drift; thus an appropriate choice for guiding the estimate towards the true state of the drone. Furthermore, it is more fitting than other alternatives such as GPS signals, with two key advantages. The first is that it can provide a higher resolution of the pose, since it is in a much smaller volume and therefore fitting the arena. The second advantage is the fact that it will work no matter where the actual arena is located, such as inside buildings where GPS signals are not available.

The fiducial markers were considered the most appropriate for this project, as they are simple to implement and have a great deal of benefits. Given the time frame to complete this project, more advanced algorithms such as edge detection with Sobel filter was not chosen due to their increased complexity. Such feature extraction could provide other benefits such as less constraints on the mechanical design, but may be more computationally expensive. Since the future implementation of the system does have real time requirements, the fiducial markers have a significant advantage. Furthermore, the availability of pose estimation algorithms through resources such as the OpenCV library facilitated an efficient prototyping phase and enabled substantial progress compared to development without libraries. Rotation matrices and rotation vectors have been used together throughout the project, as they are suited for the task. Other representations such as quaternions and Euler angles are ambiguous, as they have multiple mathematical representations of the same orientation. Algorithms to combat this may be less computationally effective and would require more work. The rotation vector was the default representation from the OpenCV library; conversion between the vector and matrix was simple with the `Rodrigues()` method.

## 6.2 Intel RealSense D435 Setup

### 6.2.1 Camera Information

It proved beneficial to extract the camera information early on, as it is important to establish a good foundation of the vision system prior to state estimation. Initially the intrinsics were extracted through the D400 Dynamic Calibration Tool (as mentioned in section 4.5.2) as it was considered adequate at the time. Eventually the cameras distortion model name was desired, which needed setup of the SDK. In hindsight that also included the desired intrinsics (and much more) for other resolutions as well, so the previous method (Dynamic Calibration Tool) proved relatively redundant. Still, it worked as a double check for the focal lengths and the principal point for the RGB module. In the future, it would be recommended to build the SDK right away to provide a majority of the relevant information in one place. The extrinsic rotations and translations were not used in the current project, but have been presented in case it is needed for future work. The rotations closely resemble the identity matrix, which is as expected since the modules face the same direction. The small difference between the listed matrices and the identity matrix may be due to the depth cameras being slightly tilted for capturing the same volume. The translation vectors' elements have similar values with opposite signs for each transformation, as expected since they go between the same modules. The slight difference between the extrinsic transformations may also be due to

rounding errors from the calculations provided by the SDK.

## 6.2.2 Camera Readings and Configurations

Only the configuration combinations which were specified in the datasheet worked for the camera, as expected. Unsupported formats may be due to limited bandwidth or hardware limitations, but this is acceptable as the supported formats were sufficient for the desired system.

### 6.2.2.1 Color Stream

The formats `bgr8` and `rgb8` display a colored image using a standard where each color is represented with a byte, or 8 bits (from 0 to  $2^8 - 1 = 255$ ). The difference between the two is the order of the colors, which is important to represent the colors correctly. OpenCV has been using the `bgr` format from its early stages, due to it being the most common format for camera manufacturers and software providers [70]. The versions with an added "a" letter to them also provide an alpha channel. The alpha parameter represents opacity, perhaps for using multiple frames to create a composite image. As just single frames were captured with solid colors, the alpha effect is unnoticeable and therefore unnecessary use of additional bits.

The greyscale formats `y8`, `y16` and the packed YUV format `yuyv` gave various results. Prior to testing, the frames were expected to look like the `y16` stream shown in figure 5.6. It was later observed that despite being listed in the terminal from the SDK, the `y16` format is in fact not listed as supported formats for the Intel RealSense D435 camera in the datasheet [24]. According to the Linux Kernel Archives, `yuyv` is also referred to as `yuy2`, which is listed in the datasheet [20] [24]. These formats are packed YUV formats, storing luma (Y) and the blue chroma (U) and red chroma (V) components [20]. There can be multiple explanations behind the artefacts on the `y8` and `yuyv` frames, such as the formats being interpreted incorrectly by the software. As this was not a central part of the project, more resources were not allocated towards research for this.

### 6.2.2.2 Infrared Stream

For the infrared stream, both the greyscale formats `y8` and `y16` worked as expected. The structured light from the infrared projector is clearly visible on both greyscale frames (figure 5.8 and 5.9), whilst not being visible to the naked eye in real life. Usually humans can only see light from the electromagnetic spectrum between 380 and 700 nanometers [105], where the laser wavelength of the IR projector on the camera is  $850 \pm 10$  [nm] at room temperature [24]. Furthermore, the projected dots are amplified on the reflective balls on the shell from the motion capture project as seen in figures 5.8 and 5.9.<sup>1</sup> This gave a great indicator for how well the projector worked, establishing relevant knowledge in case it would be needed later in the project.

### 6.2.2.3 Depth Stream

The depth frame captured live through the display windows button may be darker compared to using the script due to the program interpreting the format differently. In order to visualise the depth in the flat image, the colormap was added. Different maps were tested, but the favorite was the `COLORMAP_TURBO` due to the more dynamic and intuitive colors. The scaling factor alpha (not to be confused with the transparency alpha from the color stream) was ultimately set to 0.3 in the testing phase, as that value gave the most fitting range (around a half to one meter) for the color mapping.

---

<sup>1</sup>The parts and drones from previous projects have been handed out for research and development. The drone shell in the figure is from the module based drone project in 2018 [12].

#### 6.2.2.4 Final Configurations

In the end, the configurations for the resolution and framerate were set to be equal for the three streams, to simplify future processes if they were to be used together. In retrospect, since only the RGB camera was used in the end, a higher resolution may have been suitable; especially given the Kalman filter's ability to give an increased rate of the state estimate by updating based on other sensors and the state space model. This property of the state estimator, along with the fact that the depth reading was incorrectly implemented (discussed in section 6.3.6), was realized quite late into the project. Therefore, the resolution of the camera was not increased, to be used in tandem with the other streams and to provide a higher framerate. Furthermore, if the machine vision algorithm is desired to provide even better resolution and rate, a simpler camera without colors and global shutter may be advised. As far as this project goes, the D435 is suited for the application, both because of the specifications and the provision of more information. Even though the pose estimate from the fiducial markers only utilize the greyscale image, the project is in the future envisioned to have a HMI with the ability to enable extraction of the raw depth and color frames for the RL model, amongst other things.

### 6.3 Machine Vision Development

#### 6.3.1 Marker Detection

The simple marker detection served as a great starting point, since the pose estimation from the markers is based on the homography from this detection algorithm. As only one marker was utilized later in the project, the ID display proved redundant.

#### 6.3.2 Intrinsic Parameters

To keep the report organized, the SDK setup was introduced prior to the extraction described in section 4.5.2, though it was done at a later time. As mentioned in section 6.2.1, the SDK may be advised to use instead as it provides more information overall, varying from the format configurations to intrinsic parameters and distortion model.

#### 6.3.3 Marker Pose Estimate

The distortion is specified in the datasheet to be less than or equal to 1.5% [24]. This is considered to be insignificant as no issues were experienced when using the camera, and is backed up by research stating that up to 2-3% is unnoticeable by vision systems [51]. Also it was observed a negligible distortion when using the `undistort()` method for the resulting camera calibration, in figure 5.15 and 5.16. Furthermore, this was reflected by the results of comparing the estimated and measured distances being satisfactory without the distortion coefficients.

#### 6.3.4 Camera Calibration

The OpenCV calibration yielded the best scale, which may be due to the calibration images being taken with a more fitting dynamic range for the specific project than the Dynamic Calibration Tool. However, the optical center was better with the Dynamic Calibration Tool, which may be due to the more standardized approach. Therefore the group found it fitting to use a combination, tailored for the application. It should also be noted that the screenshot in figure 5.14 is for demonstration purposes, and is not from the presented calibration results. Taking the image with one hand while holding the calibration pattern resulted in poor detection, hence the bottom text.

#### 6.3.5 Manual Measurement Test

Though manual measurements may not be a desired methodology for validating the pose estimate, it helped determine the calibration parameters. The actual pose validation needs a more accurate and reproducible reference characteristic, hence the test designed later. The size of the marker was

not increased further, to prevent unnecessary limitations for the mechanical design of the drone airframe. The various marker dictionaries does not affect the pose estimate, as it is determined by the homography of the four corner coordinates which are the same regardless of dictionary.

### 6.3.6 Depth Readings and Combined with Marker Pose Estimate

The depth readings performed sufficiently with a good depth estimate outside the arena, and the implementation of an average value proved to streamline the reading process. A low-pass filter was considered to give a less noisy signal, but due to the requirement of a fast response of the system it was not implemented. Improved accuracy may also be achieved with a higher resolution, though originally thought undesired due to it being on the expense of the framerate. Reading only the depth alone prior to combining it with the marker readings was done to isolate the source of potential errors.

In hindsight it was realized that the implementation of the depth reading at the markers pixel position was incorrect. The depth readings were as expected, but the pixel position of the marker's center was not the same in the depth and color frames. This was observed later upon further inspection, and is most likely due to the lack of transformation between the D435 modules, since the RGB camera has a different POV to the depth module. This could have been fixed in the future, but as the project focused on the marker estimate, it was not used further. The ability to extract the depth image in its entirety to the RL model is still intended as a functionality, which could be turned on and off in the HMI. Therefore, the depth parts of the pre-arena script (appendix G.1.7) were kept as commands for the final machine vision script for future development, as seen in appendix G.1.11.

### 6.3.7 Marker Dictionary Test

The different dictionaries mostly reflected the expected results, with some exceptions. The most notable was how poor the MIP dictionary was with many false positive frames, despite its increased complexity. The 6x6 ArUco and AprilTag 36h11 dictionaries were expected to have longer computation times than the simpler ones, but it may be due to their location in the recording. As it is a negligible amount and not central for the decision, it was not taken into account. Additionally the estimate was worse at longer distances, most likely due to the reduced resolution. This was, as mentioned in section 6.2.2.4, a compromise made to operate with higher frame rate, though it was realised late into the project that higher resolution may be appropriate since the Kalman filter can give a state estimate with much higher rate despite the measurement update's lower rate. Otherwise, a new camera could be considered in the future for a better resolution with higher rate. Various depth cameras were researched, though most with better resolution and desired frame rate may not be considered within an appropriate budget. Other intel cameras were investigated, though the current D435 camera proved to be the most fitting for this application with the ideal depth range [23]. The calculated computation time and frames with data were adequate metrics to indicate the most suited dictionary, as inspection did not give a clear indication of the performance for them. False detection was important to consider to prevent an incorrect pose estimate, though visual inspection was most fitting for this, as it was important to observe what happened when the detection was incorrect. As mentioned in section 5.2.7, most dictionaries only experienced false detection on the other markers on the board. Since only one marker was going to be used in the arena, this was not a critical part of the test.

### 6.3.8 Revised Machine Vision Algorithm

As the ambiguity issue was recognized, the pose estimation algorithm was changed to the "Perspective-n-Point" version. This was done to extract more information from the estimate to decide the most correct solutions. The ambiguity issue was addressed by the rotation vector planning to be chosen based on external information. A method to provide sufficient information about the drone from the depth module was discussed which would have included reading the depth at the corners of the

marker and estimating the orientation from them, and choosing the rotation vector solution closest to it. Even though the depth readings were on average quite close to the physically measured distances, there were a substantial amount of noise and therefore this algorithm was discarded, as basing the solution choice on the orientation from the IMU is likely to be less susceptible to noise. This statement is due to the orientation's sensitivity from the depth orientation estimate algorithm, since noise in the corner coordinates would affect the plane formed by them, which could easily get amplified and drastically affect the orientation estimate for the worse.

## 6.4 Pose Validation Test

### 6.4.1 Physical Problems and Solutions

As experienced, there are many factors that can affect such a complex test. It is therefore beneficial to make a test plan to keep the experiments organized and have a checklist during development. Furthermore, the development of the test was made with export of raw data in mind, to isolate the unit under test. For instance incorrect synchronization while recording could prove difficult to correct if it was done prior to extraction of data.

The designed tool consisting of a rod and cube to move the marker around proved useful when working with motion capture, as occlusions for the cameras made QTM lose tracking. The angle of the rod down onto the cube was originally made to make it simple to move around the arena since it was believed to be placed on the floor with the person performing the test from higher ground. As the current setup was with the arena on a table, the angle was a bit awkward, but not that it impacted the test too much. If more angles are desired to be covered, a future design may consider a telescope function of the rod and perhaps a mechanism to rotate the marker along some axes at the end of it to cover even more poses, such as a gyroscope.

### 6.4.2 Calibration and Setup

As shown in table 5.8, the points for each camera vary. They display the amount of times each camera saw the markers of the Qualisys calibration tool [2]. Due to the limited space of the arena, varying detected points for each camera is expected. The average residual is well under 1 [mm], which reflects the motion capture's accuracy, showing that it is suitable as a reference. Figures 5.20 and 5.21 display the working volume of the system. QTM had good coverage of the arena, though the marker had a tendency to lose tracking when moving around the arena, even with the stick for holding the marker cube. The D435 camera has good coverage of the bottom part of the arena, but poor in the upper corners. Despite this, it can still be a viable option for state estimation, as Kalman filters have the ability to update based on other sensors and the state space model.

As discussed in section 5.3.2, the translation vectors needed some extra operations to convert between the reference frames. This was due to the rotation about the y-axis, which resulted with the x and z axes being opposite, and the y-axis staying the same. This can be seen in figure 5.22, with the qualisys oriented camera frame's axes noted as  $x_{CQ}$ ,  $y_{CQ}$  and  $z_{CQ}$ , and the camera oriented frame's axes  $x_C$ ,  $y_c$  and  $z_c$ . An observant reader may notice the order of the subtraction is opposite in equation (5.12), but since the signs are accounted for after the subtraction, it does not matter. An equal, and perhaps more correct, set of operations would be to subtract it such as in (2.1):

$$\begin{aligned}\overrightarrow{C_Q D} &= \overrightarrow{Q D} - \overrightarrow{Q C} \\ &= [x_{CQ} \quad y_{CQ} \quad z_{CQ}]\end{aligned}$$

and flip the signs of the x and z elements, since they are the axes which are opposite.

$$\overrightarrow{CD} = [-x_{CQ} \quad y_{CQ} \quad -z_{CQ}]$$

### 6.4.3 Plotting Results

As demonstrated clearly around the interval from 45 to 125 seconds in figure 5.24, the marker lost tracking when moving outside the camera. This was predicted by reading the datasheet of the D435 camera, as the FoV yielded the frustum shown in figure 5.21.

As shown quite clearly in figure 5.25, the position estimate has more error when further away from the camera in z-direction. As the marker becomes smaller in the image when further away, this behaviour is expected, and may have been improved by increasing the resolution of the camera. Likewise, the estimate is slightly worse when the marker is further away from the optical axis, though much less impactful than the z-error. The difference plots in figure 5.36 are more substantial, due to the more extreme rotations. In the future, a supervisory system is proposed to control the drone based on the state, such as a shutdown signal being sent when it exceeds certain thresholds for height, velocity and angle. Due to the sensitive nature of the drone, even small angles result in significant lateral movement. Therefore such rotations may be considered as a condition for turning off the drone. However, if a high responsiveness of the drone is desired with rapid movement, such angles may be less suited for turning off the drone. Future estimator algorithms may still be able to give an update for the state of the drone with the desired accuracy and rate even with large rotations, as they can estimate the state based on other information than the camera. Since the drone can not fly in those orientations for long, it will either turn off due to other conditions or rotate back to smaller angles and once again get camera measurements to guide the state estimate back on track.

In all the rotation matrix plots, a pattern emerged, namely the fact that elements 1,2,4,5 and 9 are much more stable than element 3,6,7 and 8. This demonstrates the ambiguity issue where the marker's pose is difficult to estimate. When inspecting the livestream with the axes displayed on the image, the rotation made the z-axis appear like it was flipped whilst it in reality was only rotated a smaller angle, around 30 degrees. This is reflected in the elements of the matrix plots, as the stable plots correspond to the x and y components of the x and y axes, and the z-component of the z-axis; which all are similar for both orientations as shown in figure 2.16.

As mentioned in section 4.7.3, some of the figures were plotted with the closest solution to the QTM measurements. For a complete experience of the attitude estimation, the angle between the closest solution and the QTM orientation was calculated. Even when choosing the closest solution, unacceptably large values were observed. Therefore, the machine vision system may be less suited towards measurements of all attitude parameters; *though there is a silver lining*. When projecting the axes onto the xy-plane and estimating the azimuth angle, the estimate is much more acceptable. Furthermore, the probability functions for the roll and pitch could let the machine vision give measurements of them as well, as they would simply get weighted much lower in the filter algorithm.

To show off the capabilities of the system, plots were made with the azimuth angle difference between the QTM measurement and the closest solution, and as shown in figure 5.34, this angle is significantly lower; even in figure 5.45 which was from the recording with much more extreme rotations. Therefore, it can be deducted that the machine vision system can be suited for measuring the azimuth angle. Furthermore, to illustrate the increased performance when choosing the closest rotation vector, the three xy-projected azimuth angle plots were made for all tests. As illustrated in both the results from test #2 (figures 5.31, 5.32 and 5.33) and test #3 (figures 5.42, 5.43 and 5.44), the performance improved when choosing the closest solution.

As seen in the XY-projection plots, the azimuth angle from the x-axis is not the same as the y-axis. This is expected, since they are  $90^\circ$  rotated from each other. The simplest plots for demonstrating this are the XY-projection plots from test #1 in appendix E.3.2 (figures E.19, E.20 and E.21) and test #4 in appendix E.3.5 (figures E.52, E.53 and E.54), since they have not been applied any trigonometric wrapping as shown in table 5.10.

#### 6.4.4 Translation Distribution Fitting

As mentioned in section 4.7.4, test #3 was omitted due to the high noise in the signals and as previously stated since it tracked an unnaturally high rotation. As shown in figure 5.46 and the other plots in section 5.3.4, the size of the plotted points were decreased to make it easier to visualize the various densities, as larger markers made the confidence intervals seem incorrect. The confidence intervals plotted in figures 5.47 to 5.49 correspond to  $\mu \pm 1\sigma$ , also known as the "one-sigma-range". This is part of the empirical rule, which states that  $\simeq 68.27\%$  of the values lie inside one standard deviation  $\sigma$  of the expected value  $\mu$ . The distributions could also have been plotted vertically on top of the points, but the confidence interval can make it easier to visualise how many of the points are inside the expected intervals.

As seen in figures 5.47 to 5.49, the confidence intervals vary depending on the distances. To make the stochastic prediction of the state estimator more correct, the standard deviations for each distribution were parameterized to later make covariance matrices which vary depending on the position of the drone. Although higher order polynomials may fit the data more closely, the chosen polynomials were of first and second degree. Given the values in table 5.11, the  $R^2$  values show that the x,y and z models account for 71.83%, 67.69% and 96.66% of the total variation about the mean, respectively. The RMSE values show that the residuals are quite small, indicating that the polynomials are useful for prediction. Note the value for the z distribution's polynomial being larger than the others, as the axes in figure 5.52 are more zoomed out than the axes of figures 5.50 and 5.51. As expected, the standard deviation for the x and y dimensions (in figures 5.50 and 5.51) have higher values when further from the middle. As stated previously, the performance was expected to become worse when the marker was further from the optical axis; therefore the polynomials for these distributions were chosen as second degree. One may consider the camera's pose estimate in the x and y axes to be modeled with the Von Mises Distribution, which is a normal distribution wrapped around the unit circle with parameters for the location and concentration; this approach has been utilized for other projects such as causal motion segmentation in moving camera videos by Bideau And Miller [13]. As the Kalman filter uses the normal distribution, the second degree polynomials were used to express the standard deviation as function of their respective positions, to construct parameterized covariance matrices for the filter. As shown in figure 5.51, the outer most two points have been omitted for the curve fitting. This was done since the performance was only expected to become worse when the marker is further from the center.

Shown in figure 5.46 there are points more spread out further from the center, but since the standard deviation quantizes the spread from the mean it gave the impression of lower spread; therefore it can be considered appropriate to not include them, as the mean is non-zero there. These non-zero mean values of the intervals can be observed in the figures 5.47 and 5.48, as the middle of the confidence intervals are above zero. These systematic errors may be due to factors like lighting conditions or other bias sources in the camera, but the most likely reason may be due to the placement of the body frame in QTM. It was done by translation from the center and rotation from the plane formed by the holes to the surface of the cube. This rotation may have caused a slight misalignment, since the reflective markers for motion capture were placed outside the cube, resulting with the plane formed by the reflective balls not being parallel to the plane formed by the holes. Furthermore, the chosen production method for making the cube was done with 3D printing and glue for the screws, which also may have caused misalignment. If even better performance of the marker system is desired, a new cube is advised to be produced in the future, possibly machined with threads for the screws in a light metal like aluminium. As the current test had the error bias in the range  $\mu \in [0, 0.03] \text{ [m]}$ , it was considered appropriate to simply continue with the design of the translation Kalman filter to demonstrate the capability of the vision system.

#### 6.4.5 Position Kalman Filter: 1 DOF

As seen in the state space equations in (4.5b), the filter does not utilize a feed-through matrix  $D$ . As the filter used the simulated high rate sensor as input  $u$  and yielded satisfactory results without

it, the feed-through matrix  $D$  was not included.

The one-dimensional Kalman filter was shown with the results from the pose validation test #1 data in a smaller region of the time range, since it displays a good range of the movement without any significant errors. As shown in figures 5.54 and 5.55, the estimate became better with the filter than direct measurements from the pose estimate of the marker with the machine vision system alone. This is further emphasised with the data shown in tables 5.12 and 5.13, as both the mean and standard deviation values are lower with the filter. One exception to this is shown in the latter table for test #2, where the values are higher, especially for the standard deviation. This is an error due to the linear interpolation, where the interpolated reference values were calculated between a large gap due to the cube moving too fast for the Qualisys system. The true value moved towards the camera and away in an arc instead of linearly away from the camera, which resulted in an incorrect calculation. This is shown when inspecting the error's peak in figure E.64 and the other values in figure E.63 around 120 to 130 seconds in the recording. This was the primary reason for zooming into the region of interest in section 5.3.5. A similar situation arose for test #4, seen in figures E.70 and E.69. Although not as protruding, similar peaks in the error are observed in other plots where interpolation was not as applicable. Overshoot was experienced when using the "spline" option instead of linear interpolation, hence the latter choice. When the measurements  $y$  were not detected, the filter relied only on the input  $u$ ; this lead to a drift since it was basing new estimates on old data. This can be shown in figures E.58 and E.57 around 50 to 55 seconds in, where the filter drifts away from the true value and accumulates error until the measurements are back. This is one of the main reasons for the visual pose estimation, as a lower rate sensor (in this case the vision system, other systems use for example GPS/GNSS) can provide more consistent data over time despite the noise, and guide the estimator towards the true value.

#### 6.4.6 Position Kalman Filter: 3 DOF

As both the 1D and 3D filters are based on the same data, the z results were omitted from the latter. Figures E.77 and E.78 illustrate the fact that the camera lost tracking prior to the smaller time range, since it went outside the covered volume shown in figure 5.21 (this is also discussed in section 6.4.3); this was the reason for the smaller time range together with the fact that it is simpler to read than a zoomed out plot. As seen in figures 5.60 and 5.61, the estimate of the x and y distances were quite accurate, but have a systematic error. This is most likely due to the bias discussed in section 6.4.4. Bias compensation can be accounted for with an error estimate in estimators (further discussed in section 6.4.7), but this bias is likely to be due to the slightly incorrect reference values discussed in section 6.4.4.

The covariance matrices  $R$  and  $Q$  for the 3D filter were made with only the variances on the diagonal. This means the noise only takes the variance into account and not the covariance from the various parameters in each matrix affecting one another. Further complexity was not developed, as the filter is a simple demonstration of the visual system's performance and the group did not have time for further R&D regarding the state estimator. More information regarding this project's involvement with a state estimator and discussion about further work is found in section 6.4.7.

#### 6.4.7 Further Work: State Estimator

This years project did not have time to also make a state estimator for the RL arena, as it would require too much time. Development of a working state estimator for the physical system requires a substantial amount of work, as there are many things that need to work together. First off, the filters introduced in this report have only been for translation, whilst the physical implementation would be more suitable to start right away with a full state filter which would also require orientation. In fact, due to the non-linear nature of orientations, a regular Kalman filter would not suffice; nonlinear estimators such as the EKF or sigma-point filters could be utilized. Furthermore, to decompose the acceleration vectors correctly for subtracting the gravity measurement and interpret the data from sensors mounted on the quadcopter for the estimator, the reference frame

needs to be converted from the body frame to the desired frame (could be camera centered, arena actuator centered, etc.). To do the transformation from the body frame, this frame's orientation needs to be known. This can be done by calculating the new orientation from the previous iteration with kinematics for rotational motion, which again would require proper knowledge of the previous orientation, dating back to a known orientation from the start position. The reason for the advice of making a full state filter with orientation for physical implementation, was that the translational filter needs IMU data in the right frame, hence the orientation requirement, initialization sequence and a more advanced filter algorithm to combat the non-linearity. The bias in the IMU would also need to be accounted for, such as the leveling algorithm proposed by Kruithof and Egeland [64].

Future work with development of a state estimator may consider starting with inspiration from one of the error dynamics filter variations with either the feedforward configuration in figure 6.1 or the feedback configuration in figure 6.2, or a full state filter [37]. More advanced filter algorithms could also yield more computationally efficient implementations which are faster, thus more desirable for the planned real-time applications of the drone RL arena.

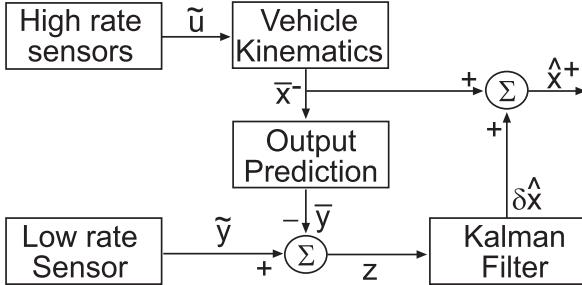


Figure 6.1: Feedforward complementary filter implementation [37]

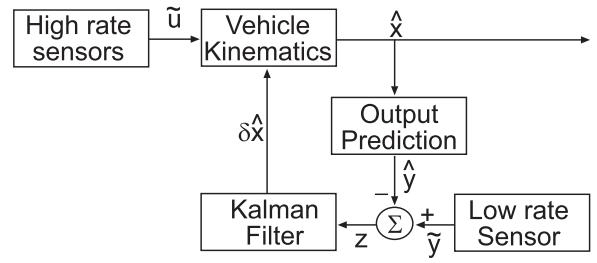


Figure 6.2: Feedback complementary filter implementation [37]

## 6.5 Server

The original purpose of the server, as mentioned in section 1.3, was to be able to transfer messages from one process to another with real-time capabilities, the server achieved this well as can be seen in the results section 5.4.1.

For the real time capabilities, the results section 5.4.2 shows the delay between when a message was expected and when a message arrived which was on average 0.361[ms]. For a real-time system, a latency of 6-20[ms] is considered sufficient, so the results were positive although improvements can still be made [65]. This may be improved by changing the way the offset is calculated for when a message is expected. The initial message was only received at the next sending of the message as can be seen in the results, this may be caused due to a mistiming of starting the programs, however a loss of a message did not occur for the remainder of the test so this was considered unimportant as in this point of time in the arena no movements of the drone would have started.

In hindsight, the creation of the server was not the best use of time and choosing ROS2 for this may have been a more time-effective solution, however, due to the heavy system requirements for ROS2, the server has an advantage as it can be used on lighter single-board computers or possibly ported to micro controllers which support WIFI such as certain ESP32 or STM32 chips. This can allow the drone to be mass produced for much cheaper as opposed to using a CM4 as the current design uses.

### 6.5.1 Further development

The current server system allows for a trivial implementation of the supervisory system. Due to the use of queues, a function could be made that takes the output of the RL drone motor control

messages along with the position output from the state estimator, reads the state estimation and changes the kill-switch boolean on the drone control message if the drone meets the criteria to shutdown before passing the messages on to their respective queues for sending. This could possibly be run in a thread to allow for simultaneous use of the HMI. Unfortunately this was not implemented due to a lack of time and because fully functioning implementations of the related clients were incorporated late into the project.

The HMI was partially implemented in the server code although not to the desired degree. The current system allows the user to input values into the program to start or stop the system, but it was intended for this to have more usage such as changing which processes communicate with each other during runtime, changing options for what to provide for the RL model such as for the camera to send raw images or change reference frames. It was also desired to be able to save the various options to a file which could be loaded again for future runs of the program.

With the use of protocol buffer messages, it was initially assumed that it would be easy to separate the type of message from the encoded string, however it was later found out that this task was not as easy as expected and would require a very large change to all of the messages which there was not enough time to do as small changes to the protocol buffers requires changing many sections of the code. A method to separate messages could be done by using the OneOf field types [117]. Due to the way the messages ended up being set up, the end of transfer messages was set as the decimal value '4' which corresponds to the ASCII End-of-Transmission character [1]. Although this could be changed in the future for a protocol buffer message as then certain options could be passed as well. With this method, messages for re-synchronizing the timers or changing sending interval times could also be sent after starting the system, but in the current implementation, if a different type of protocol buffer message was sent than expected then it would throw an error when parsing it. Due to not being able to send multiple message types, the development of the HMI got hindered.

The method used for synchronising the timer was also found to have some negatives, the best method of synchronising the timers would have been to send the server's timer based on the epoch time rather than a point in time based on the monotonic time, however using the epoch time was not used as it was feared that issues would be created based on one of the client's setup of the wall clock, so the current synchronising setup was used instead. With using the monotonic time, the sending intervals would become un-synchronised if the interval was not a multiple of the delay time used in synchronising the timers between the client and the server, as shown in figure 4.15, therefore if a client wanted to change the interval time, the whole synchronisation process would have to be redone. Therefore the current implementation only allows the interval times to be set during the initial setup and not at runtime due to the issues with the protocol buffer messages discussed earlier.

In the current implementation of the server, the synchronisation sleep time was hard-coded to 100[ms], however the sleep time could be sent in the reply message from the server at the start of the synchronisation process. This time could then be used for the sleep function to get a timer set for the desired sending or receiving interval.

For the python client, due to the use of multiprocesses rather than threads, it was not possible to use the same threading system as with the C++ programs because sharing a socket between multiprocesses is unnecessarily complex for the task, therefore a method for sending and receiving messages was put into a single loop which uses the same sending interval [11]. Despite this, a method in the python client of checking if the expected sending and receiving intervals were equal or not was not implemented due to a lack of time, and also because this may cause an issue with custom made receiving and sending methods using the socket.

For the C++ server and client, the usage of threads and the way messages are passed in the server and general system may be controversial, often it is considered bad practice to share sockets between threads however it was considered fitting in this scenario as the threads only serve one purpose (sending or receiving) and does not impede on the other thread which it shares a socket with.

## 6.6 Actuator

### 6.6.1 Absolute encoder

While testing the absolute encoder, it was found that small movements of the actuator system changed the readings of the encoder by a large amount, this means that if the arena were to be disassembled and reassembled again, then the readings from the sensor would change from what is previously was, this was shown in the figure 5.64. This is an issue as the purpose of the absolute encoder was to provide a way of always knowing the rotation of the actuator system, and the changes in readings would affect the sensors ability to achieve this. When the arena was screwed down, shaking the arena system had no affect on the readings however, which means that the drone crashing into it would not affect the readings.

This error likely occurred due to the magnet shifting above the encoder which causes a change in rotational readings, during the creation of the encoder system this error was not expected but it ended up being a major issue as can be seen from the results from testing the encoder by reading the values during a rotation test with the azimuth motor in Section 5.5.3. As mentioned in the results, due to the bad tolerances of the rotating vertical actuator section, as the azimuth motor rotated it, the entire section was shaking which likely created a lot of bad readings from the absolute encoder. When the motor shifted from one rotation to the other, then entire section switched position which caused a big change in the way the encoder read the magnet, this can be seen in Figure 5.65 where after the 1 second stop, the motor changes the rotation direction. An image of this effect can be see in figure 6.3, which shows how the magnet shifts to a different position based on the encoder on a rotation change.

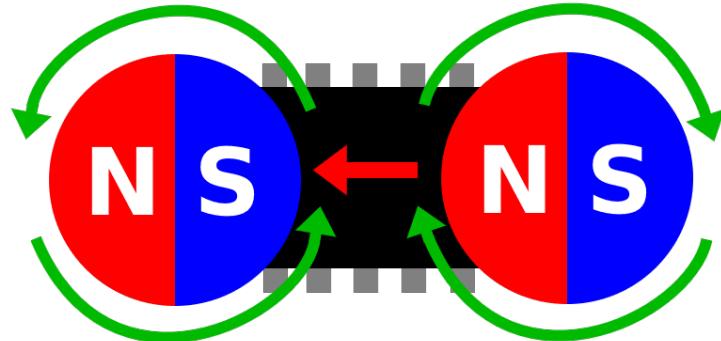


Figure 6.3: Image showing how the position of the magnet shifts above the encoder

This issue may be possible to be fixed with tighter tolerances on the arena components as well as replacing the thrust bearing with a rolling-element bearing which would keep the rotating section held in place, alternatively a combined filter could be created from the absolute encoder and the encoder attached to the motor however this would be significantly more complex [77]. Unfortunately, this test was done very late into the project so there was not enough time to fix these issues, and due to a lack of connectors for the servo motor, the motor encoder was not able to be connected and only the wires for steering the motor itself was able to be plugged in.

Because of the height increase due to the absolute encoder, the power lead screw was also supposed to be made 20 [mm] shorter to prevent the top stick cap from hitting the lead screw, however there was not enough time to make this change due to the late implementation of the absolute encoder.

## 6.6.2 Communication

### 6.6.2.1 USB communication

The group ran into issues with the programming file size for the microcontroller and difficulty to send serial data through C++ code, as mentioned, the simple test code was 4 [kB] of flash memory whereas the ATMEGA32u4 had 32 [kB] of flash memory, therefore the group decided that it might not have enough space for the entire code of the actuator so an upgrade to a microchip such as the AT90USB1287 with 128 [kB] of flash memory would be required [8] [9]. The group therefore decided to explore other alternatives first.

### 6.6.2.2 Server

Because so much time was spent on coding the server, there was not enough time to create the ROS2 programs that move the actuator system motors. Making a program which converts messages from the server to ROS2 messages was also not completed due to the lack of time, however protocol buffer messages and ROS2 messages have many similarities which would make this a non-trivial task.

### 6.6.3 PCB design

All the tests within the section 5.5 were done with a connection to the PCB before the raspberry PI, showing that the PCB worked as intended without any major issues requiring it to be reordered. The choice of 0.07 [mm] rather than the standard 0.035 [mm] copper thickness ended up working well and saved a significant amount of space on the PCB despite costing more. The previous group which worked on the drone arena project used a 4 layer PCB, however a 2 layer PCB was desired as it is easier to note the tracing of the PCBs physically as all the traces are visible [14].

Ideally the flyback diode should be placed close to the inductive load which in the case of the arena would be on the wire to the QI charger, however it was decided to be placed on the PCB as it was noted on physical inspection that the QI charger seemed to have internal circuitry to protect it from flyback, and because the charger would only be switched on and off every couple of minutes giving good time for heat to dissipate if heating had occurred.

The choice of contacts could also have been more standard, most of the contacts were based on the requirements of the wiring such as the current, rather than what was already in use. This meant that a lot of soldering to new contacts had to happen even though the previously used contacts could have been utilized. Pre-crimped wires could have also been purchased to save a lot of time. For connecting the PCB to the raspberry pi, it may have been a better alternative to use a ribbon cable that connects to all of the GPIO pins of the raspberry PI rather than using wires with a 1 pin socket and selectively choosing the GPIO pin to connect it to. This method was considered after the PCB was designed, however it may also create issues for expandability in the future as there would be no free pins to connect to.

After ordering the PCB, it was found that some elements were forgotten. From following the previous group's connection of the motor drivers, the enable pin of the DRV8825 was set to be permanently grounded which meant that the stepper motor was always on, this ended up causing the driver and stepper motor to become very hot while the system was powered, even if the motor was being driven or not. This would however be a very simple fix as the RPI input connector J9 had an unused pin labeled with "X" which could be used for the enable pin of the stepper motor. The resistor for the 12[V] power LED was chosen as a  $3.3[k\Omega]$  resistor, however this was primarily chosen due to a misreading from the previous group's PCB which can be shown in Figure 6.4 along with the lack of information on the LED from the manufacturer of the rack. The LED ended up lighting up however it was very dim, and it should be considered to be replaced with a lower value such as a  $400 - 1200[\Omega]$  resistor which would give the LED  $30-10[mA]$  respectively, which is the recommended current for lighting LEDs [107]. The new iteration of the PCB used exclusively standard labels, such as 3K3 for this resistor label, which can be shown in the top left of the soldered PCB in Figure 4.20.



Figure 6.4: Image showing resistor label with "330OHM" [14]

Another oversight was that a resistor for the QI charger was forgotten, the QI charger would currently short ground and 12V if the QI charger did not have a receiver, therefore using Ohm's law and  $P = u \cdot I$  to get an output of 5[A], a 60[W] 2.4[ $\Omega$ ] resistor would need to be placed in series before the QI connector. This component was ordered but was not fully implemented. It was also desired for some of the VIA placements to be doubled up in areas where a lot of current would pass through a single VIA, such as the QI ground VIA and other motor grounding VIAs, although it was calculated that this should not be an issue. The VIAs for the motor drivers did not become excessively hot during testing of the motors.

Since the sockets for the motor drivers were sold out in all of the component stores, it was unfortunately not possible to re-order the newer iteration of the PCB.

## 6.7 Drone Hardware and Software

### 6.7.1 Buildroot OS

The custom operating system is shown to use 41 out of 1824[MB] of system memory as seen in Appendix 5.1. The CM4 hardware is vastly overpowered for this use case and could therefore be replaced with a less powerful unit. If additional functionality is required, the tutorial in appendix E.5.5 can assist in generating a template for a more feature rich operating system.

### 6.7.2 IMU

With the BMI088 being used in the shuttleboard form while connected to a CM4 in the IO board, as shown in the graph 5.66, the accelerometer worked as expected and gave a value of 1g in the respective axis. However, the gyroscope seemed to be broken as the signal would periodically cut out and give only readings of 0, this can be seen in the graph 5.67. The periodic readings of 0 was output from every register of the gyroscope, and attempting a self test as specified in the BMI088 datasheet gave a reading that the gyroscope was functioning as expected, therefore the cause of this was likely external, possibly due to damage to the gyroscope or from noise in one of the signals from the raspberry PI, however the exact cause could not be determined [15]. Despite this, the gyroscope seemed to give correct readings when not outputting 0, although it was not fully verifiable as the signal could not be integrated properly to give an angle reading in this broken state.

## 6.8 Mechanical Drone Design

### 6.8.1 Physical Design Discussion

The RPi based flight controller allowed for parallel development of software and hardware, since the CM4 and RPi 4 units use the same processor. However, due to the form factor of the PCB carrying the CM4, the size of the electronics compartment is quite generous. In addition to the

large wingspan caused by the prop guard design, it is evident that the drone is quite large for a sub 250[g] assembly. If a smaller flight controller was used and mounted along the battery, the overall size and weight of the drone could be reduced by a significant amount in the future. The pre-existing flight controller was quite heavy with its approximately 25 gram weight. An optimized flight controller may be subject to discussion for future projects, as there is room for improvement both for weight and dimensions. Unused, thus redundant components such as the barometer, time of flight sensor, and manual switch inside the drone could be removed.

Care was taken to ensure the design did not feature too many complex shapes to enable a simple conversion for future vacuum forming. Though some modifications would have to occur, the general shape of the mechanical drone design should be a solid step in the right direction. The shapes that need the most modification are the propguards, as the straight angle would not be vacuum-forming friendly; a draft angle of at least 2° should be added to aid the removal from the stencil. A new L shaped lip could be added to the overlap between the top and bottom parts of the hull, around the outer contour to create good contact geometry for gluing them together. 3D-printed PP has different material properties depending on what direction forces are applied relative to the layer direction [95]. Vacuum forming from a sheet of PP would create a more homogeneous material structure.

A PLA test drone was printed to see if any unforeseen problems would occur with the form. The print assembled without any major problems. Slight fitment issues occurred between the top and bottom hull shapes due to shrinkage. This was anticipated and the top part was printed slightly larger to compensate. When printing in PP it was found that the shrinkage during cooling was not always consistent. A reason could be that some of the parts were left inside the enclosure while the machine shut down over night, letting them cool at a slower rate than parts that were removed from the enclosure while still warm.

### 6.8.2 Computational Fluid Dynamics Simulations

To verify the data from the CFD simulations, the group sought after corroborating data. In the 2022 report, the drone is reported to require approximately 50-75% throttle, however without knowledge of the throttle curve used in betaflight, this information reveals little about the drones performance. It is known that the previous drone's flight capabilities were less than optimal with a 2s battery, struggling to stay airborne in angle mode, and a reluctance to gain altitude vertically [14]. Further reading into the 2022 report reveals a graph of the drone accelerometer during flight. For a brief moment the drone is seen to have achieved a vertical acceleration of 1.25[g]. If the battery is assumed to have been fully charged at the time of recording, the acceleration matches very closely with the simulated value for full throttle. With a weight of 250[g], the drone requires

$$F = m \cdot a = 0.25[\text{kg}] \cdot 1.25 \cdot 9.81 \left[ \frac{\text{m}}{\text{s}^2} \right] = 3.07[\text{N}] \quad (6.1)$$

to match the acceleration given. The simulation result for that drone with 3800  $\left[ \frac{\text{rad}}{\text{s}} \right]$  was 2.933[N]. One possible error source is the Gemfan 3025 propeller which was too complex to model accurately with the groups current skillset. A visually accurate model was obtained on GrabCAD [19]. Since Gemfan provides no data or model of the propeller themselves, this was the best option available to simulate the current setup. If the model itself is inaccurate, the same propeller is used on both drone simulations. This way the relative performance should be reasonably accurate, even if the exact force number deviates from what an experimental approach would yield.

All four arm surfaces combined on this year's design make up 0.1488 out of 0.24[N] in total parasitic drag. Percentage wise, the new design loses more to various other factors. For example, air passing over the inside of the prop guard surface would slow down, reducing overall thrust. The simulation considers this area glossy, but after printing the surface was textured which could affect how the airflow passes over it. The 2022 drone lost substantial amounts of thrust due to its arm shape, size and proximity to the propeller. Even with the 2024's arm shape being the least aerodynamic shape

possible, the sum of forces lost to parasitic drag is

$$\frac{F_{2022}}{F_{2024}} = \frac{1.1256[N]}{0.1488[N]} = 7.565 \quad (6.2)$$

times more on the 2022 design. Further refining the arm shape on the 2024 design would make this figure even bigger. The resulting thrust to weight ratios are, assuming 250g for both drones

$$TTW_{2022} = \frac{2.933[N]}{0.25[kg] \cdot 9.81 \left[ \frac{m}{s^2} \right]} = 1.196 \quad (6.3)$$

$$TTW_{2024} = \frac{3.749[N]}{0.25[kg] \cdot 9.81 \left[ \frac{m}{s^2} \right]} = 1.523 \quad (6.4)$$

Like mentioned earlier, the interesting parts about these simulations are the relative performances more than the actual thrust values until the CFD integrity can be reliably verified.

For hovering purposes, the duct-like propguard design will perform well, but the downsides become more apparent during transverse flight trajectories. As the propeller moves through the air, the propguard hinders both in and outflow from the blades. Additionally, the large top surface area needed to room the electronics and fiducial marker does not lend itself to any high speed maneuvers or acrobatics. Considering the initial goal for the drone is learning to fly itself inside the relatively cramped drone arena space, the lack of high speed maneuverability and efficiency is not a huge concern at the present time. However, if the end goal is to perform acrobatic maneuvers, the drone would likely benefit from a smaller design.

### 6.8.3 Physical and Simulated Impact Tests

During preliminary physical testing, the 3 [mm] wide brackets securing the battery in a corner almost immediately broke after falling from more than 1 meter onto concrete floors. After the brackets were ruled out as a reasonable method to secure the battery, 3[mm] cable ties were procured and the battery remounted. The drone was reassembled using a hot glue gun for the hull and cable ties for the battery. When dropped from 5[m], the drone landed on its belly three times in a row. The first time the bottom lid partially detached. The second and third time a crack formed along the edge between the sidewall and revolved wall. It is no surprise that the failure occurred in these locations, as the stress on the bottom lid was parallel with the 3D-layer direction. The second crack is a textbook stress concentration over a sharp edge. If additional toughness is required, adding a 3D-print layer to the bottom and smoothing out the edge would be simple ways to reduce the chance of failure.

The PLA frame was originally meant to be a placeholder material until it could be replaced by a stronger material, but having survived the 5[m] drop test three times, it is difficult to argue against it. The only concern remaining for PLA is if it is a rigid enough material for the motors and propellers to have a stable platform for applying thrust. If later iterations of the drone would seek to increase the thrust to weight ratio, stiffer options might need to be considered. Using an FR-4 fiberglass frame could allow the circuit board to be embedded into the frame itself, along with providing greatly increased stiffness (Tensile strength: 320[MPa] [72]). Alternatively a carbon fiber plate could be machined for even greater stiffness (Tensile strength: 3730[MPa][73]), but being electrically conductive extra care would have taken to insulate electrical components. Both options have higher mass density than PLA's 1.24 [g/cm<sup>3</sup>] [96], with 1.9 [g/cm<sup>3</sup>] and 1.87 [g/cm<sup>3</sup>] respectively [72][73]. Despite their higher density, it would be possible to obtain a stiffer arm with a lower overall weight due to the superior material properties.

The prop guard design was not impervious to impacts and deflected more than initially anticipated, but the physical impact tests indicated that the SolidWorks simulation results were valid. Keeping the propguards from contacting the propellers during direct impacts from great heights would require a very stiff design, and is considered unnecessary as the Gemfan 3025 propeller is ductile and able to bend out of harms way if the prop guard deforms enough to make contact with the propeller. During a flight cycle in the arena the propellers of the drone will spin and could harm the drone, but a proper supervisory system is planned to stop them prior to impact to prevent unnecessary damage.

The simulated impact tests were done with both scenarios that are likely to occur during RL flight cycles in the arena, as well as unlikely worst case scenarios. As seen in sections 5.7.4.1, 5.7.4.3 and 5.7.4.4 the common scenarios are relatively harmless, as the deflection is within an accepted tolerance. The vertical drop onto the single propeller guard yielded much higher deformation due to the impact in this orientation. As the thrust is in the direction normal to the lateral movement, direct impact in this orientation is not very likely, in addition to the fact that the drone is likely to have angular momentum during flight cycles in the arena.

## 7 Conclusion

The work presented in this thesis has described the design and development solutions for making the drone reinforcement learning arena one step closer towards autonomous flight. Given the suitable advantages provided with a state estimator, the implementation of a visual pose estimation system has been developed with emphasis on evaluating its performance through comparison to a reference characteristic. The pose validation has been a priority, aiming to make the tests reproducible through production of equipment, a test plan, programs tailored for the arena tests and detailed information; facilitating future tests conducted by other researchers. As a proof of concept, a simplified state estimator based on the validation test data and analysis was simulated, demonstrating the visual system's potential. The presented analysis reflected the viability of using the vision system for x and y positions as well as the azimuth angle. If further performance from the system is desired in the future, a higher resolution could be utilized and completing the pose validation tests with a precisely machined cube for reduction of systematic error. As the project focused on making the tests reproducible, future completions of the test procedure are alleviated. A framework for communication has also been implemented in the arena, offering the crucial connections between the system's processes. The electrical system for the arena actuator has been redesigned along with the implementation of certain features for the actuator system. The thesis also included the development of a new airframe for the quadcopter, designed for resisting the many iterations of the flight controlled by reinforcement learning in the arena. The 2024 drone design achieves higher efficiency than the previous iteration. Areas of improvement for the drone could include developing a lightweight flight controller, evaluating a higher  $K_V$  motor option for a better thrust to weight ratio with the 2s battery, or reducing the electronics compartment size. The resilient drone design appears to be a promising platform for further development, providing a lightweight solution.

*This page intentionally left blank.*

# Appendices

## A Technical Drawings

Following the same structure as the thesis, the technical drawings in this appendix are in three categories. The first is the drawings related to the pose validation test and the machine vision, starting at A.1. The second part is related to the arena development and contains new models which were made in this project, starting at A.2. The third and final part is the most prominent of this appendix, as it is the main mechanical aspect of this thesis. It includes the developed parts for the quadcopter in this project, starting at A.3. As seen in section A.3, only the technical drawings made in this project are given. The remaining parts shown in the assembly in subsection A.3.1 are primarily for illustration of the physical quadcopter assembly. More information about the assembly parts without drawings are seen below in table A.1:

Table A.1: Drone Assembly Information

Part	Description
2 Cell Battery	Illustrative, found in this project's repository
Circuit Board	From the 2022 FC project [50]
Propeller	Gemfan T 3025 from GrabCAD [19]
Motor	From the 2022 arena project [14]
M2×5 Hex Socket Screw	Illustrative, found in this project's repository
M2×8 Hex Socket Screw	Illustrative, found in this project's repository

The weights shown in section A.3 were mostly measured on a Sartorius H120 toploader scale at campus. The circuit board's weight was not taken into account, as a complete design is incomplete and listed as further work. The M2 screws were too small to get an accurate reading of, so their weights were found with an online calculator [74].

8

7

6

5

4

3

2

1

F

F

E

E

D

D

C

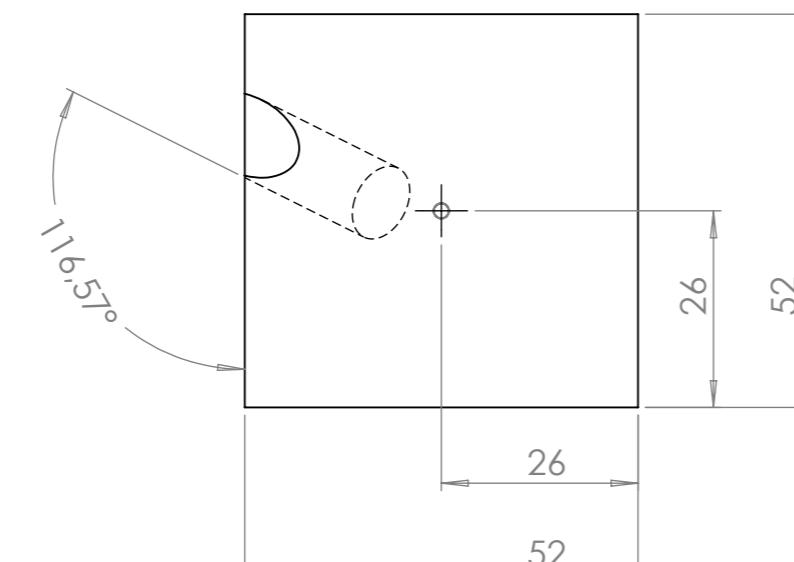
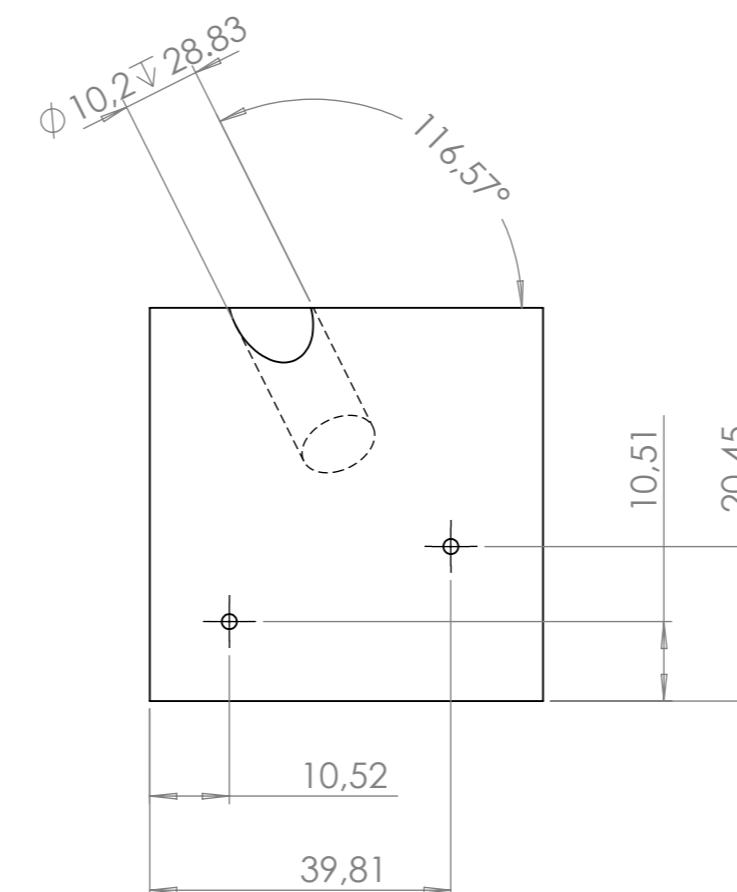
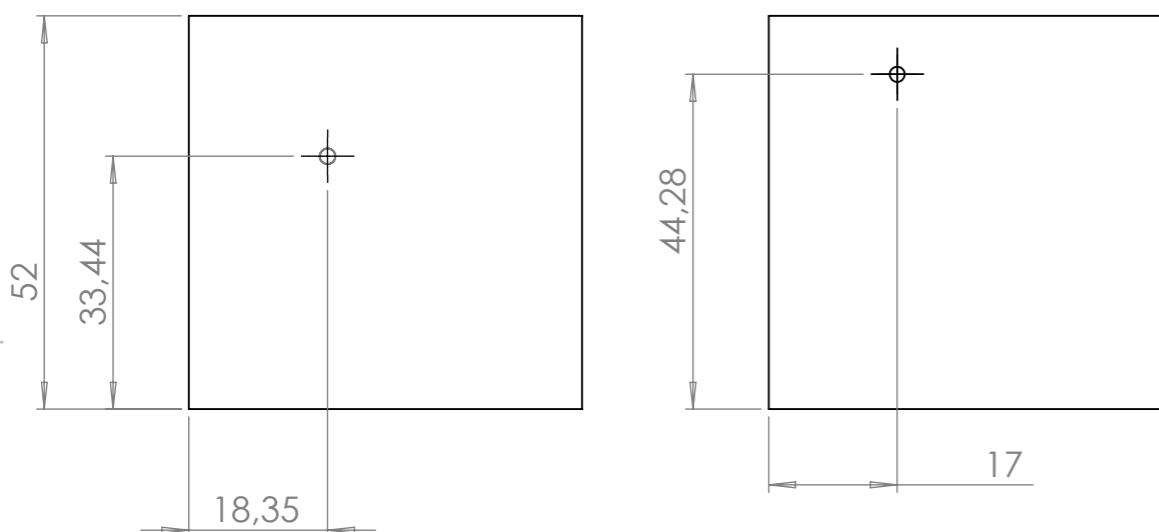
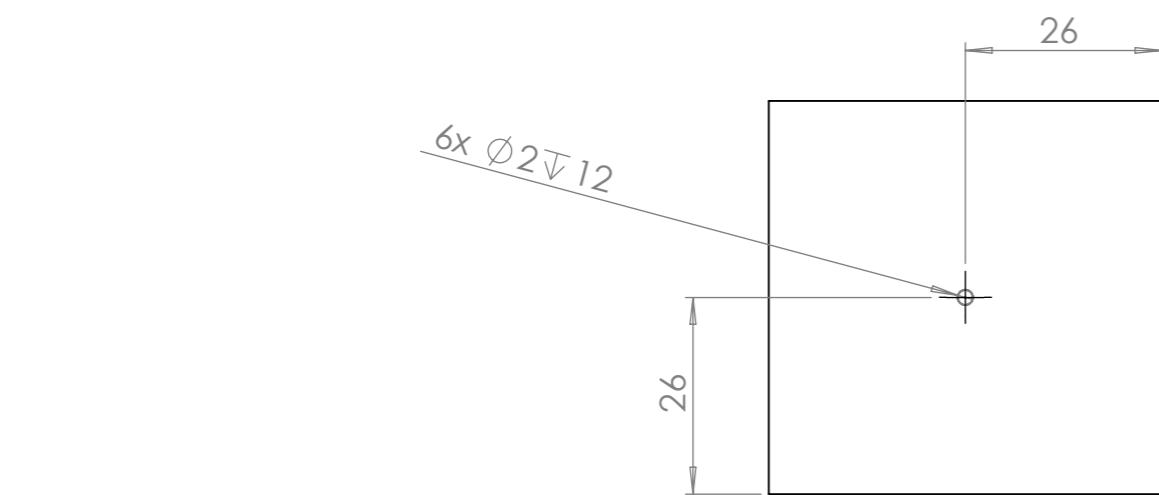
C

B

B

A

A

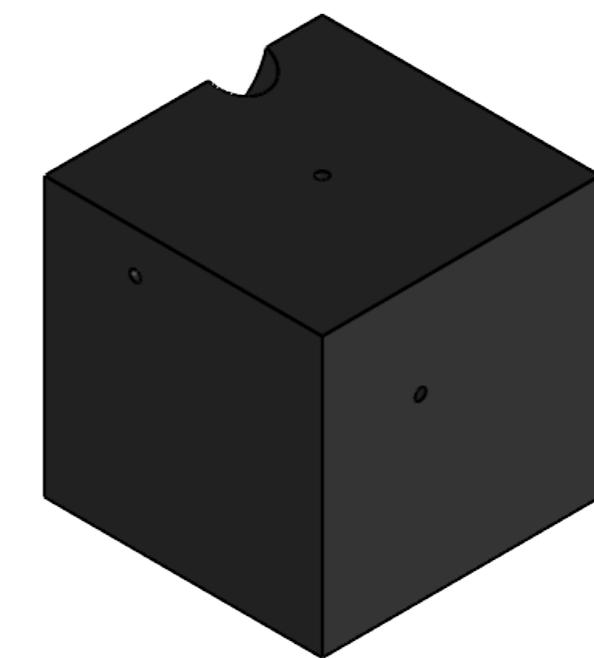


This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted:

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	19.05.2024 JLH	19.05.2024 AMLL	07.05.2024/TLS



All measurements are in mm U.N.O.				Projection:	DO NOT SCALE DRAWING		REVISION: A
DRAWN	NAME	SIGNATURE	DATE	FINISH:	University of Agder		University of Agder
CHK'D				REPLACES:			
APP'D				REPLACED BY:			
Q.A.				Specification:	Marker Cube		
				ASA			
				DWG NO.	poseTest-1		A3
				WEIGHT: 141.13g	SCALE:1:1		SHEET 1 OF 1

8

7

6

5

4

3

2

1

F

F

E

E

D

D

C

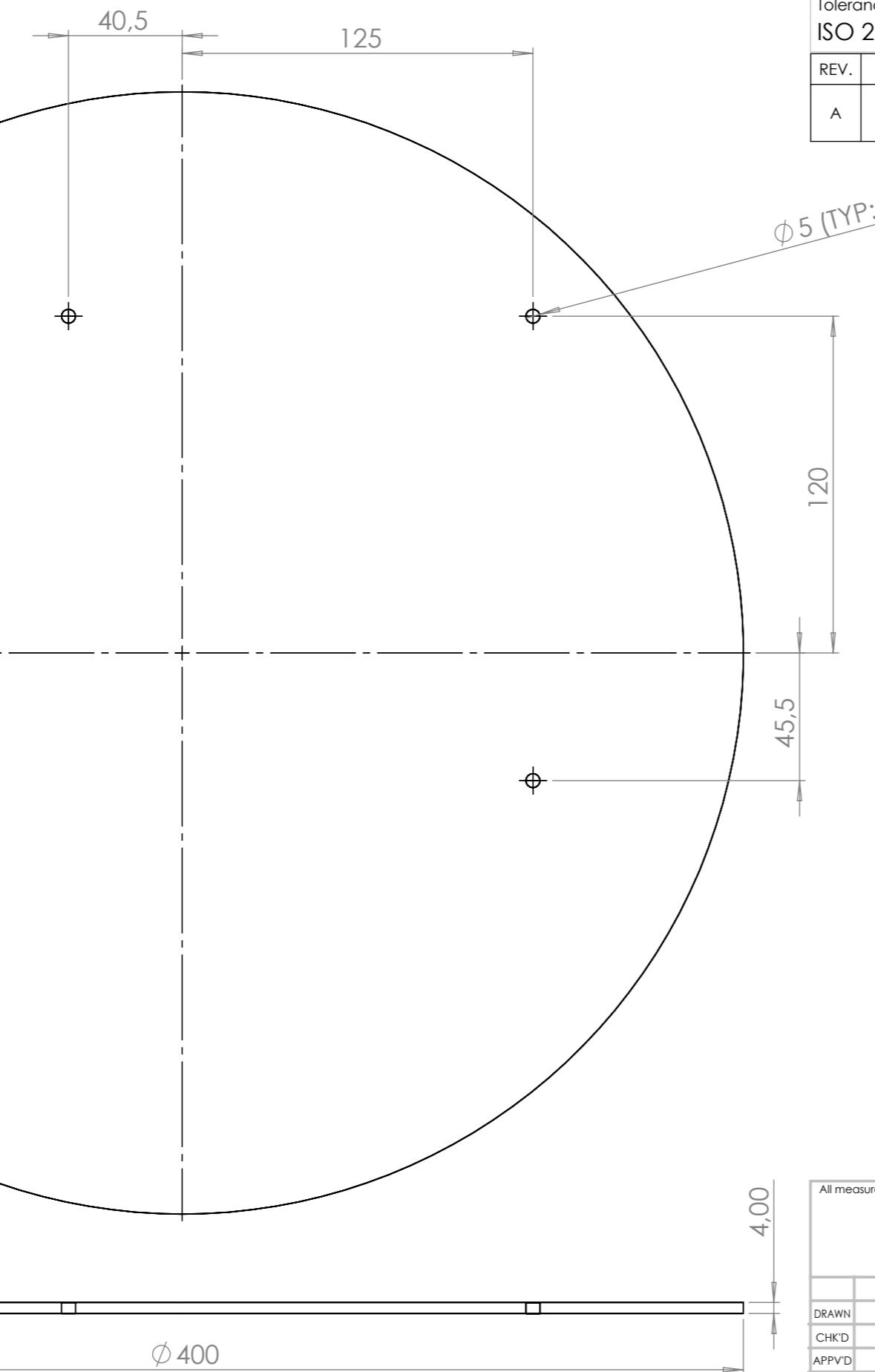
C

B

B

A

A



This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted:

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	19.05.2024 JLH	19.05.2024 AMLL	08.05.2024/TLS



All measurements are in mm U.N.O.				Projection:	DO NOT SCALE DRAWING		REVISION: A
DRAWN	NAME	SIGNATURE	DATE	FINISH:			
CHK'D				REPLACES:			
APP'D				REPLACED BY:			
QA							
				Specification:	DWG NO.		
				Acrylic Plastic	SCALE:1:2		poseTest-2
				WEIGHT: 602.90g	SHEET 1 OF 1		A3

University of Agder

Calibration Plate

8

7

6

5

4

3

2

1

F

F

E

E

D

D

C

C

B

B

A

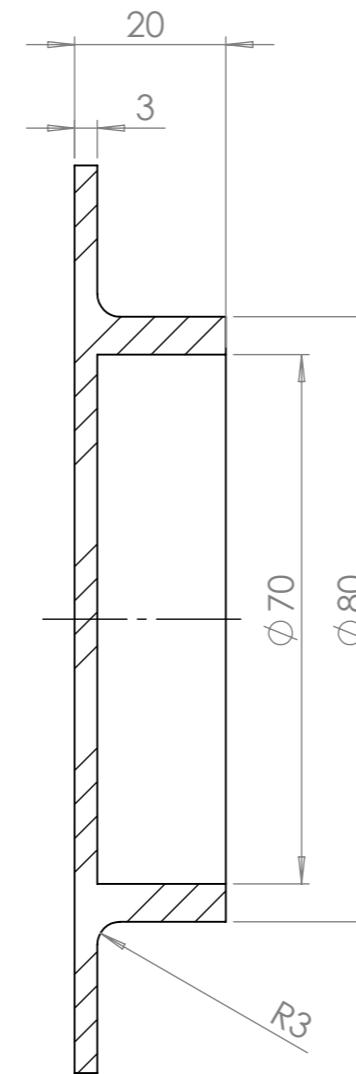
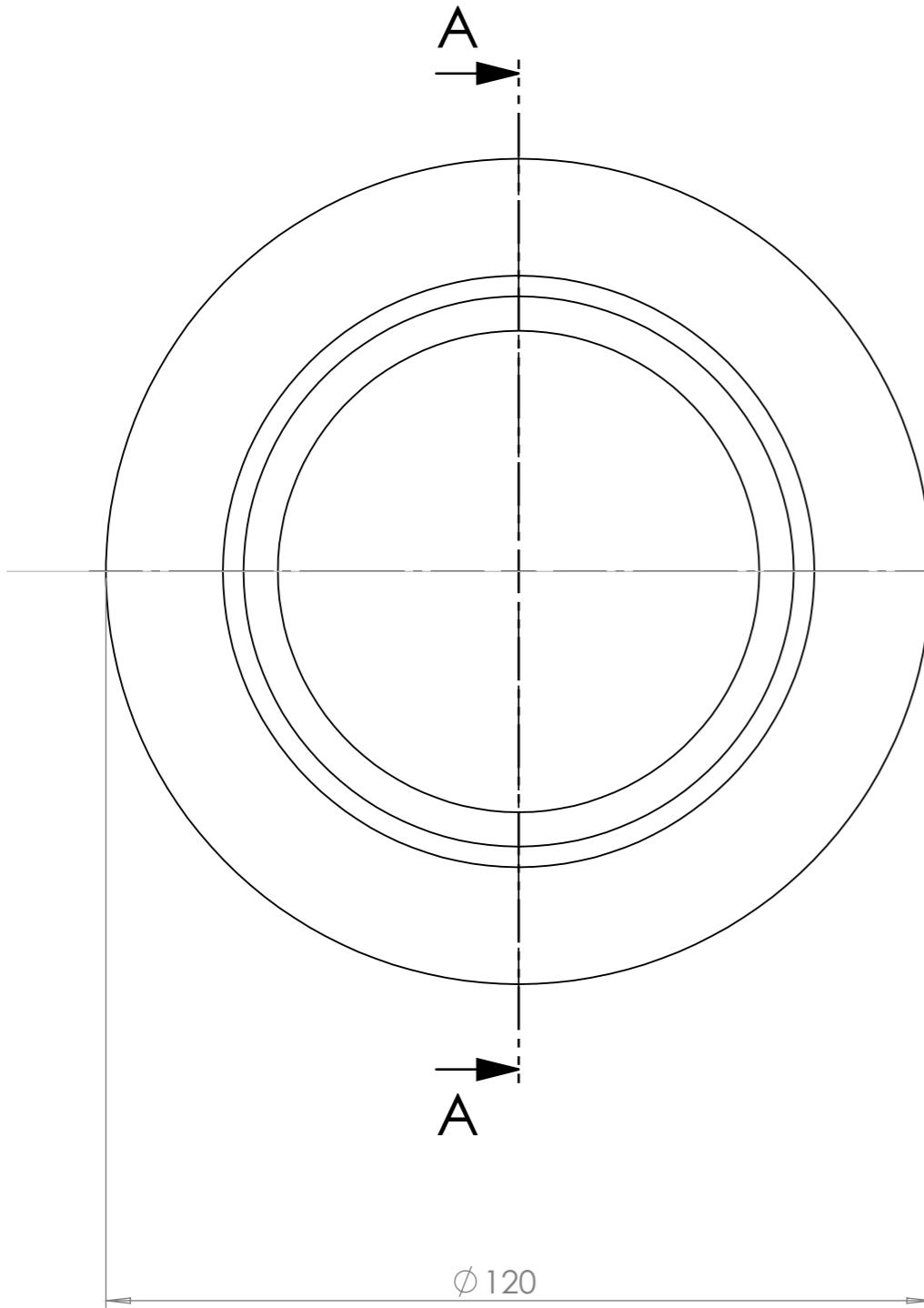
A

This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

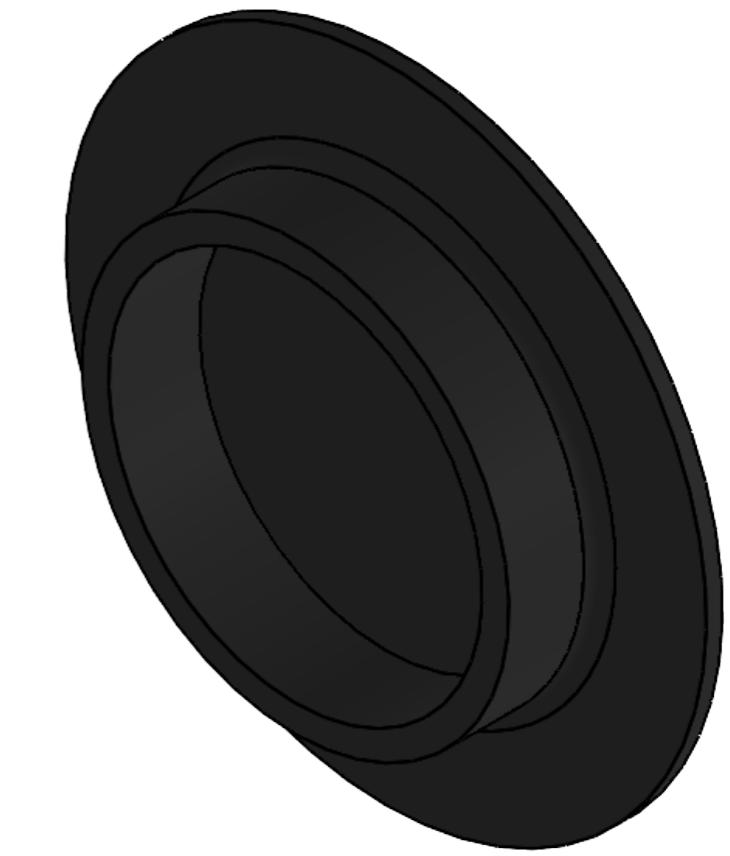
Tolerances except where noted:

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	19.05.2024 JLH	19.05.2024 AMLL	07.05.2024/TLS



SECTION A-A



All measurements are in mm U.N.O.				Projection:	DO NOT SCALE DRAWING		REVISION: A
DRAWN	NAME	SIGNATURE	DATE	FINISH:			
CHK'D				REPLACES:			
APP'D				REPLACED BY:			
Q.A.				Specification:	PLA		DWG NO.
				WEIGHT: 68.06g			SCALE:1:1
							SHEET 1 OF 1
				poseTest-3		A3	
				Actuator Bracket		University of Agder	

8

7

6

5

4

3

2

1

F

F

E

E

D

D

C

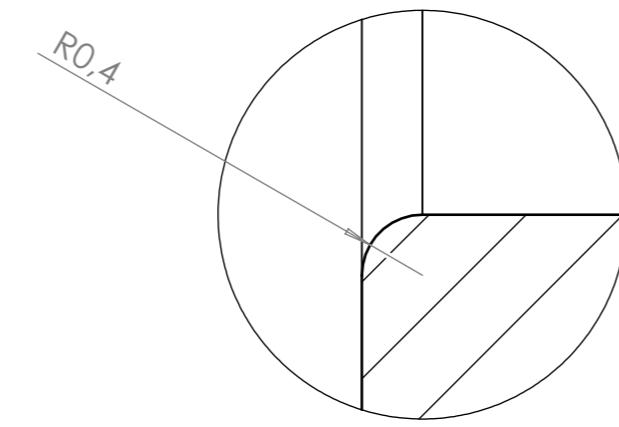
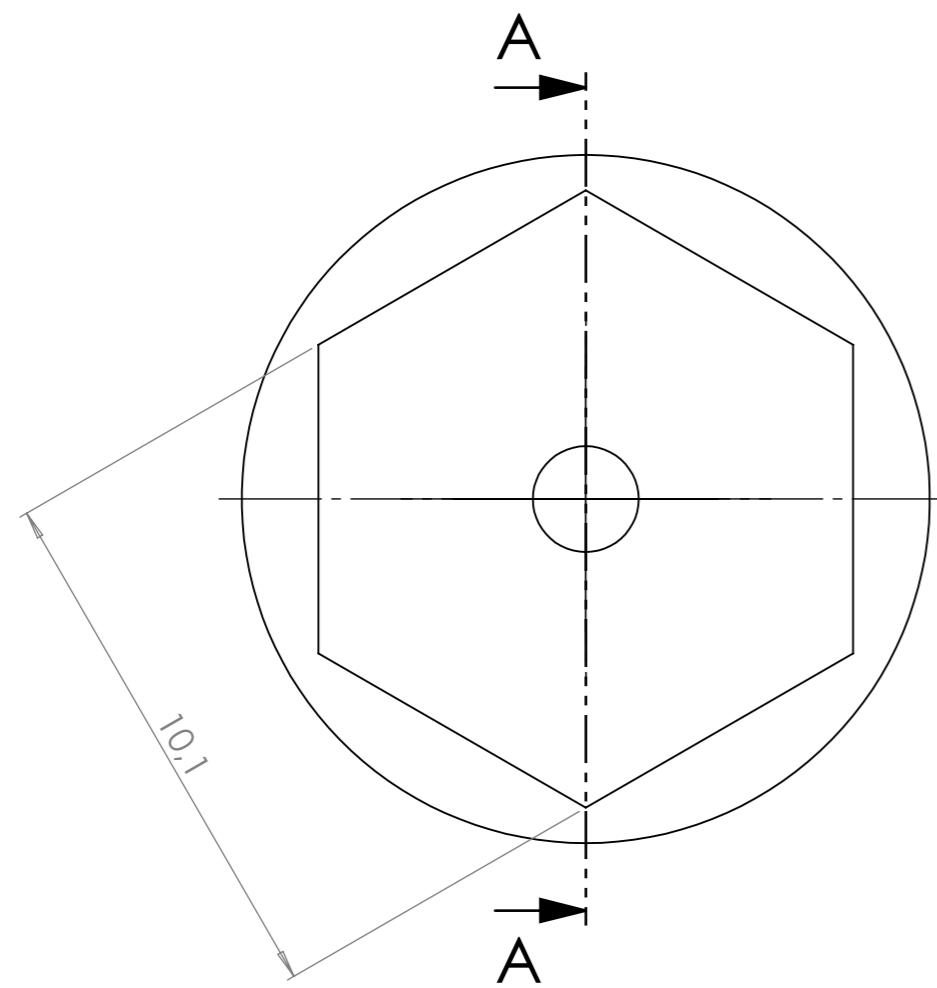
C

B

B

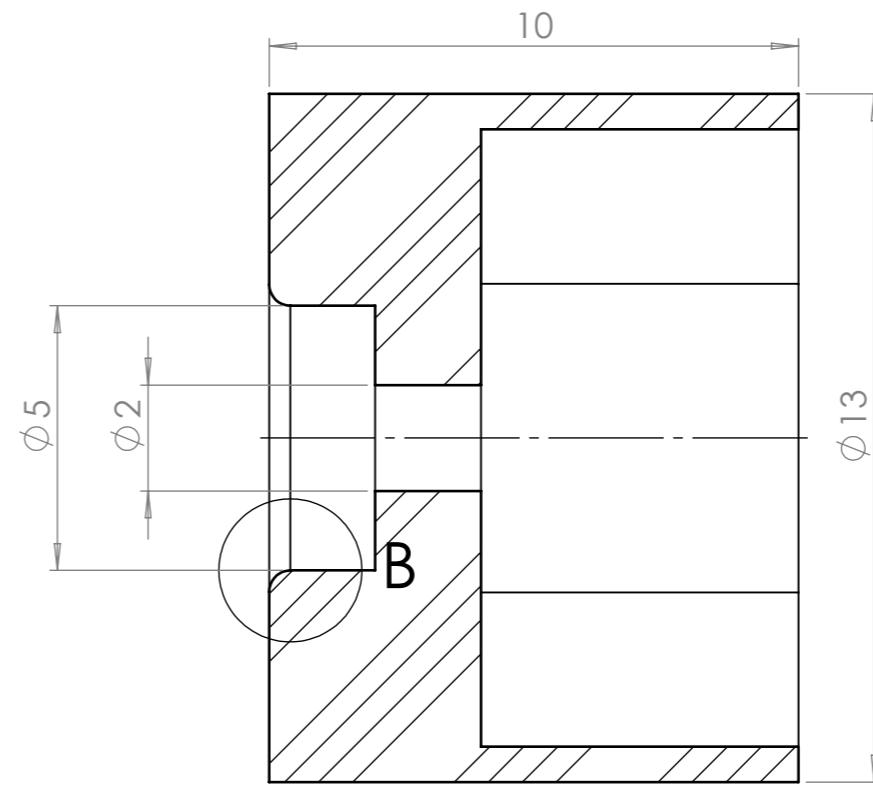
A

A

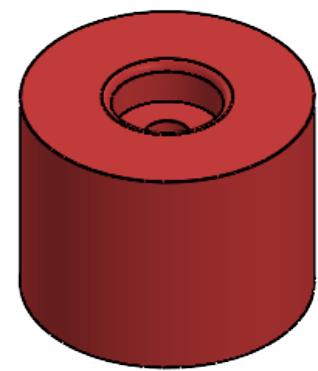


**DETAIL B**

SCALE 20 : 1



**SECTION A-A**



All measurements are in mm U.N.O.				Projection:		DO NOT SCALE DRAWING	
						REVISION: A	
DRAWN:				REPLACES:		University of Agder	
CHK'D:				REPLACED BY:			
APP'D:				Specification:	PLA	TITLE: Magnet Holder	
Q.A.				DWG NO.		arena-1	
				WEIGHT: 0.94g		A3	
				SCALE: 7:1		SHEET 1 OF 1	

8

1

1

1

1

1

1

1

F

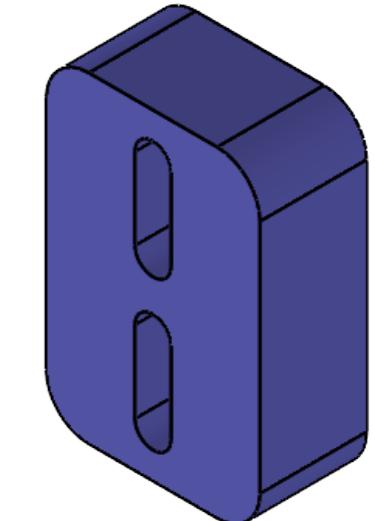
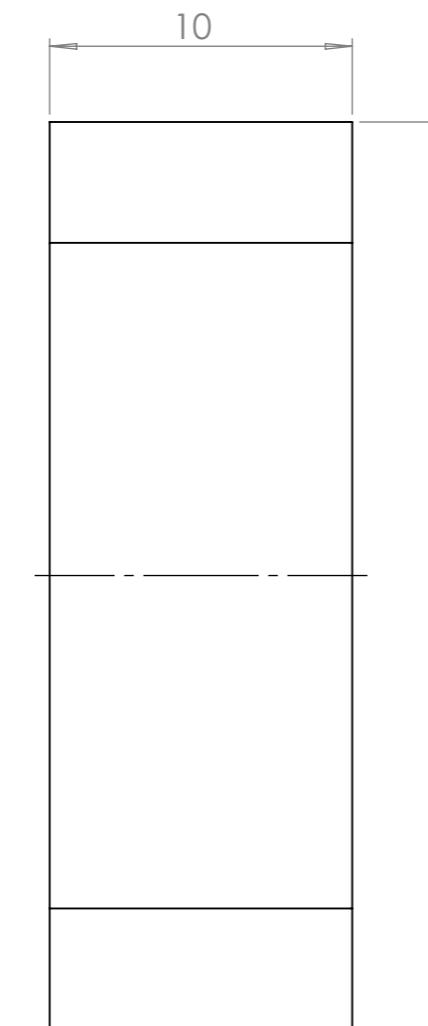
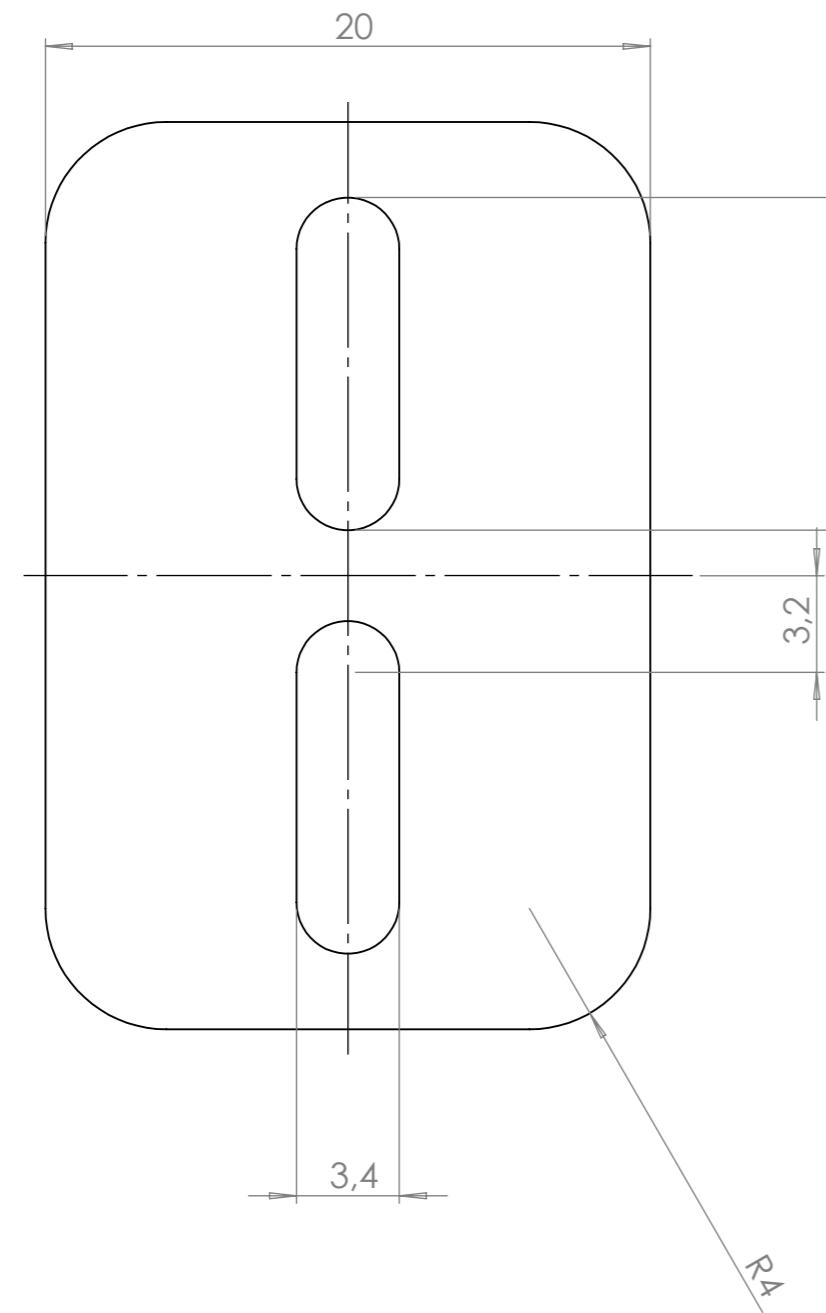
F

This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 B

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	19.05.2024 JLH	19.05.2024 AMLL	19.05.2024/TLS



All measurements are in mm U.N.O.				Projection:		DO NOT SCALE DRAWING	REVISION: A
	NAME	SIGNATURE	DATE	FINISH:		University of Agder 	
DRAWN				REPLACES:		TITLE: Button Block	
CHK'D				REPLACED BY:			
APPV'D							
Q.A.				Specification:	PLA	DWG NO.	A3
				WEIGHT: 6.46g		arena-2	
					SCALE:4:1	SHEET 1 OF 1	

8

7

6

5

4

3

2

1

F

F

E

E

D

D

C

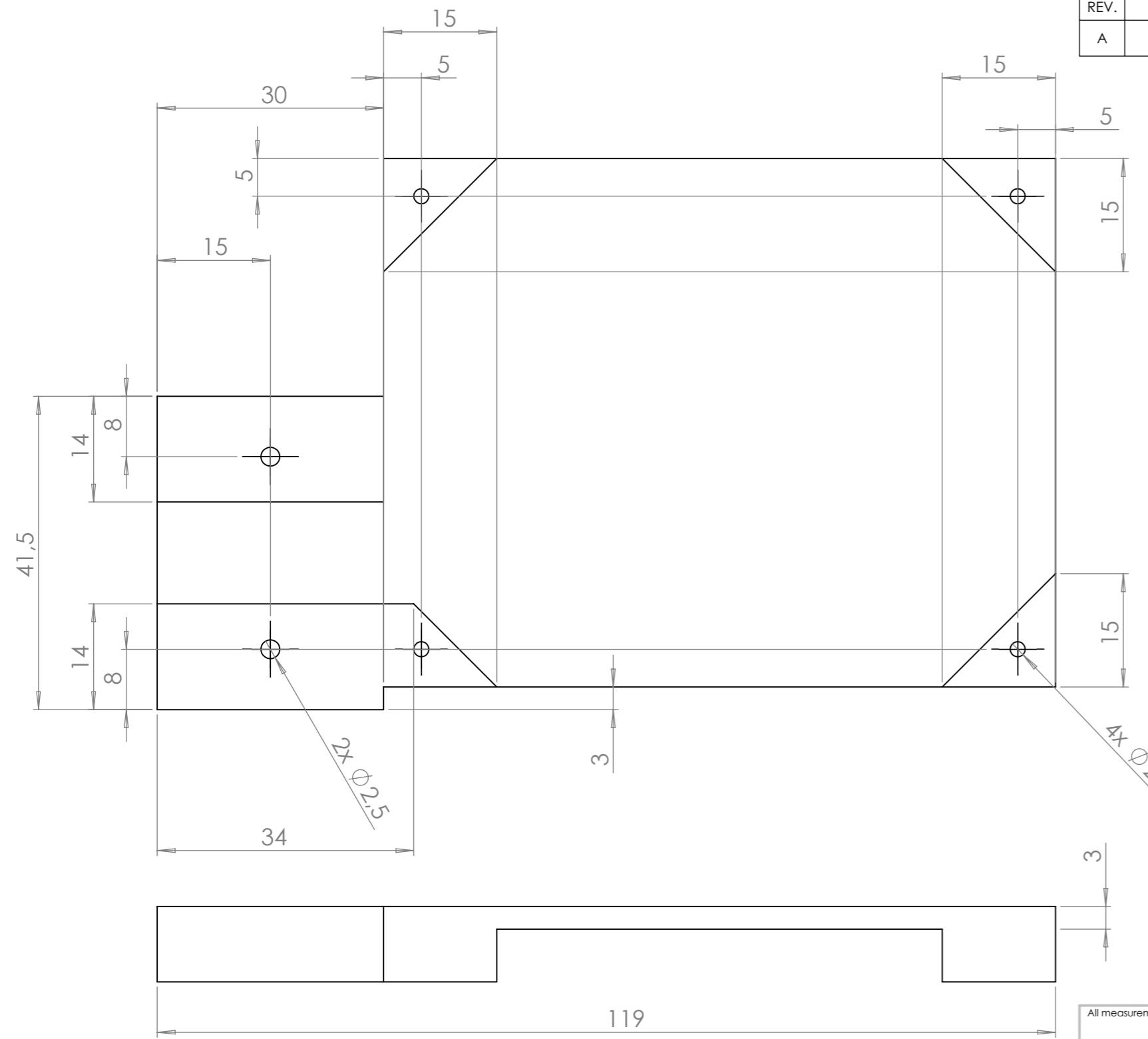
C

B

B

A

A

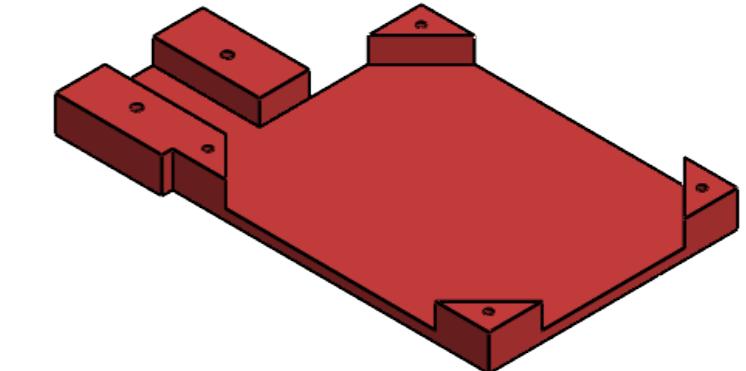


This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted:

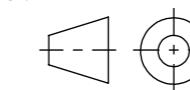
ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	19.05.2024 JLH	19.05.2024 AMLL	19.05.2024/TLS



All measurements are in mm U.N.O.

Projection:



DO NOT SCALE DRAWING

REVISION: A

University of Agder



Card Holder

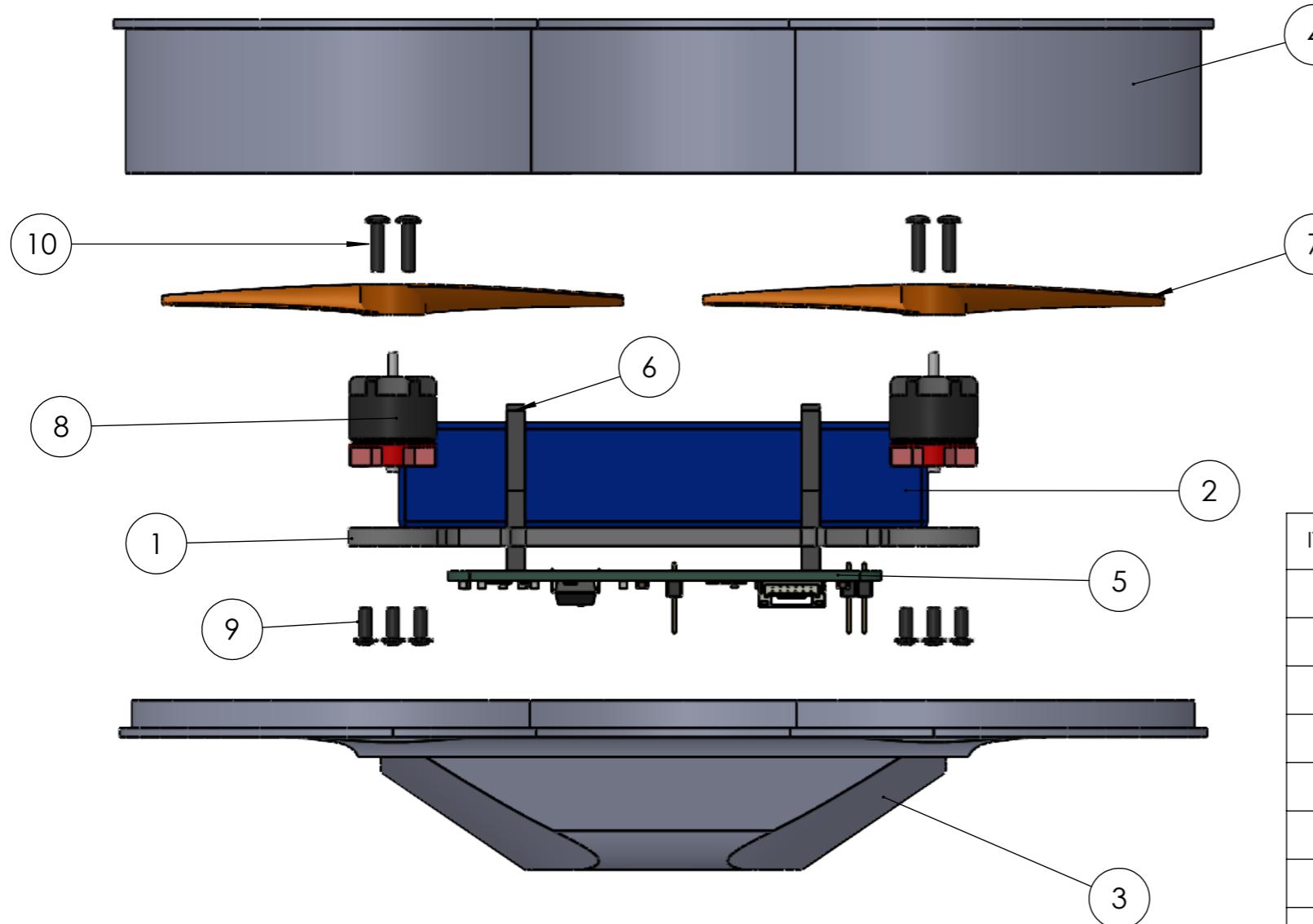
DRAWN	NAME	SIGNATURE	DATE	FINISH:	Specification: PLA	TITLE:	DWG NO. arena-3
CHK'D				REPLACES:			
APP'V'D				REPLACED BY:			
Q.A.							
				Weight: 38.97g		SCALE: 1:5:1	SHEET 1 OF 1

This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted:

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	19.05.2024 AMLL	19.05.2024 JLH	19.05.2024/TLS



ITEM	QTY.	PART NUMBER	DESCRIPTION	UNIT WEIGHT [g]
1	1	drone-3	Airframe Core	18.8
2	1	Battery	2 Cell Battery	85.5
3	1	drone-1	Airframe Bottom	17.2
4	1	drone-2	Airframe Top	45.3
5	1	circuitBoard	Circuit Board	NA/TBA
6	2	drone-4	Battery Bracket	1.1
7	4	mk3Gemfan 3025mk2	Propeller	1.0
8	4	mk3Emax_USAmk2	Motor	7.3
9	16	M2x5	Short Hexagonal Socket Screw	0.157
10	8	M2x8	Long Hexagonal Socket Screw	0.227

All measurements are in mm U.N.O.				Projection:	DO NOT SCALE DRAWING	REVISION: A
DRAWN	NAME	SIGNATURE	DATE	FINISH:	TITLE:	
CHK'D				REPLACES:		
APP'D				REPLACED BY:		
Q.A.				Specification:	DWG NO.	
					drone-0	A3
				WEIGHT: 206.928g	SCALE:1:1	SHEET 1 OF 1

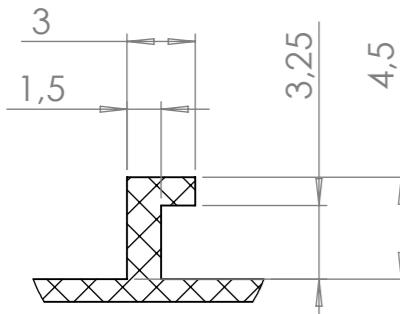
8 7 6 5 4 3 2 1

This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted:

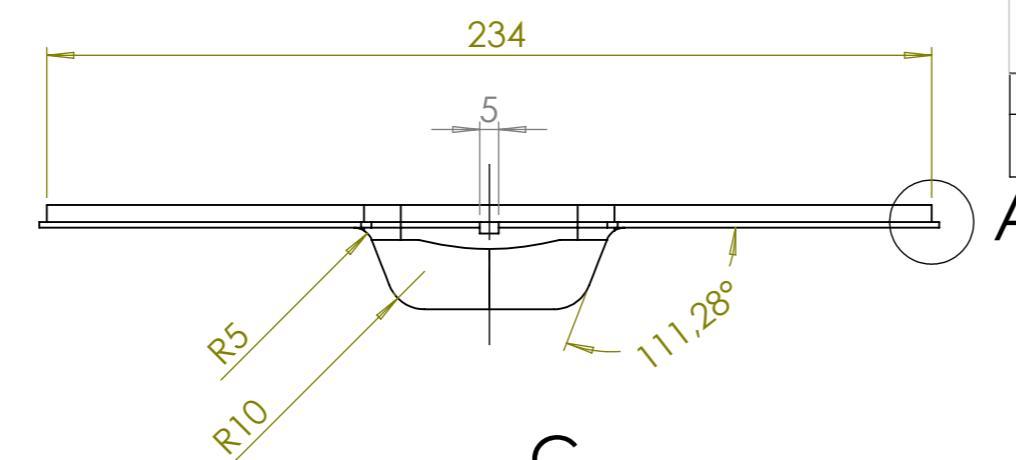
ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	18.05.2024 AMLL	18.05.2024 JLH	16.05.2024/TLS

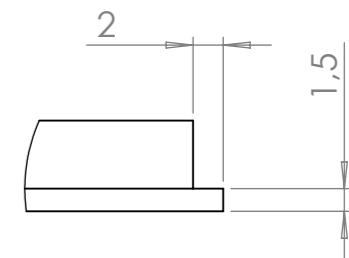


DETAIL D

SCALE 3 : 1



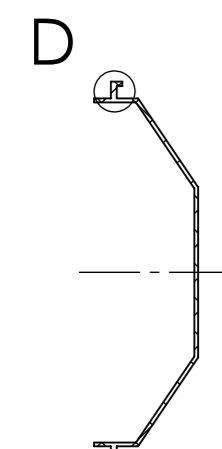
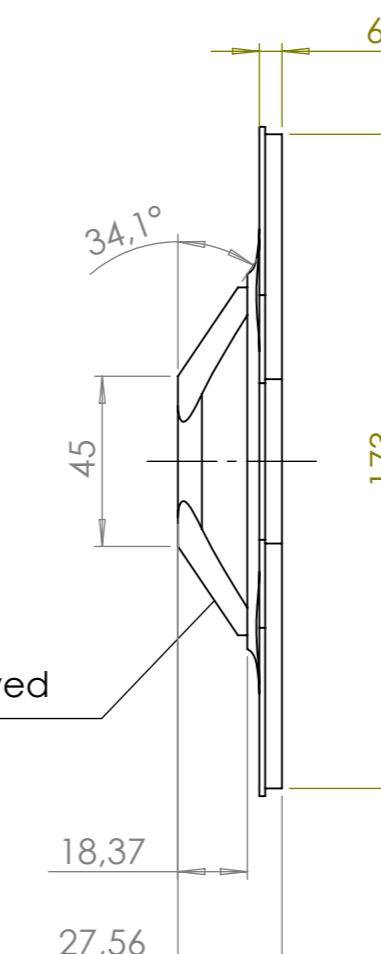
A



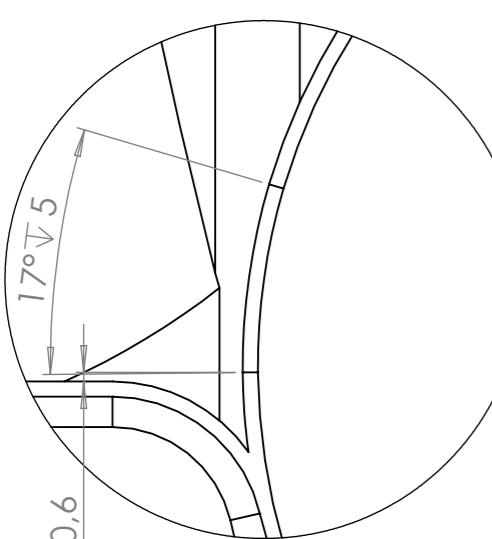
DETAIL A

SCALE 2 : 1

Swept boss around outer contour



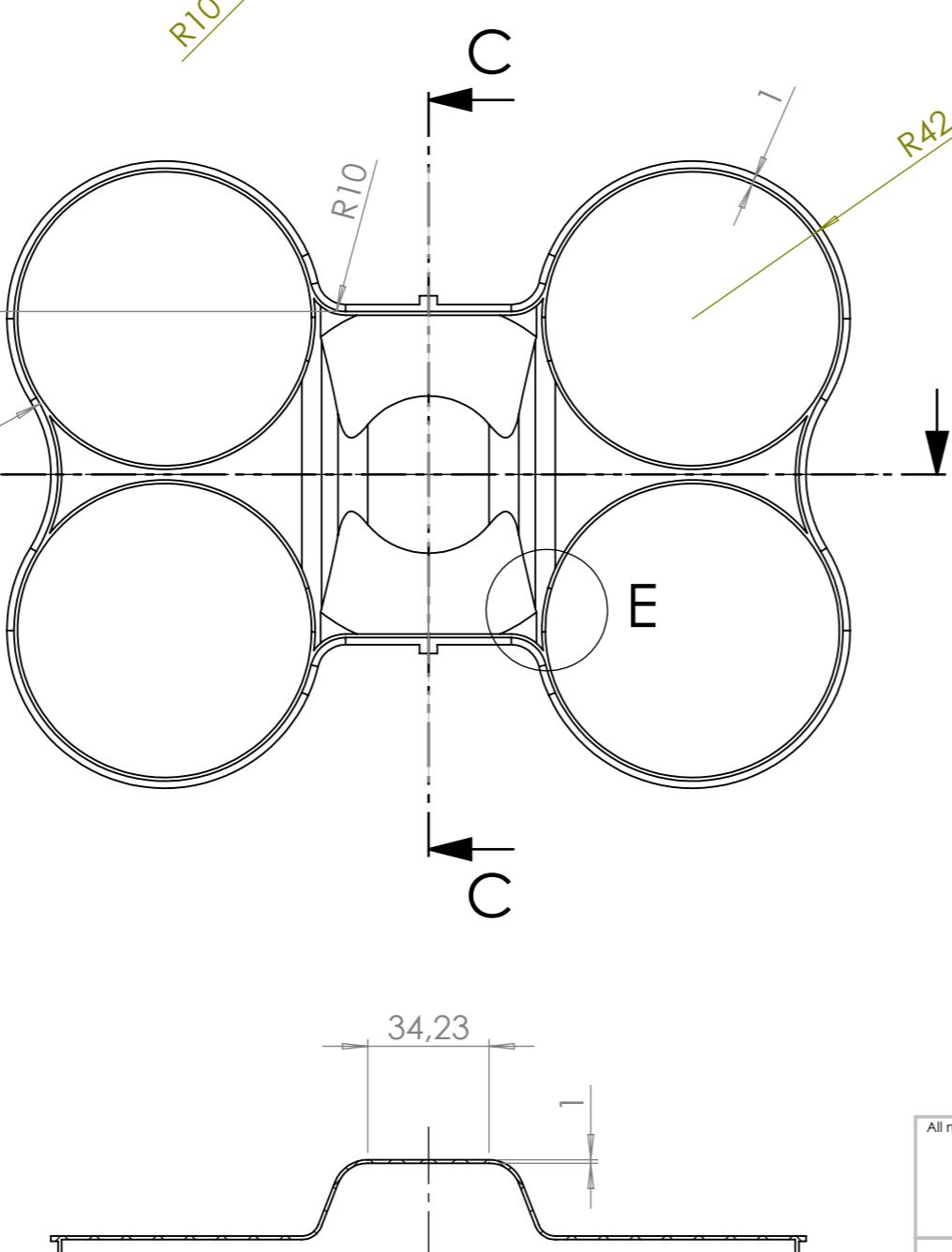
SECTION C-C



DETAIL E

SCALE 2 : 1

Arm slots



SECTION B-B

Hollowed airframe

All measurements are in mm U.N.O.				Projection:	DO NOT SCALE DRAWING		REVISION: A
DRAWN	NAME	SIGNATURE	DATE	FINISH:			
CHK'D				REPLACES:			
APP'D				REPLACED BY:			
QA				Specification:			
				P-Filament 721		DWG NO.	
				WEIGHT: 17.2g		SCALE: 1:2	
				SHEET 1 OF 1			

University of Agder 

Airframe Bottom

drone-1 A3

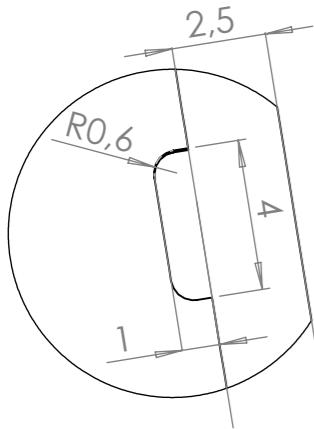
8 7 6 5 4 3 2 1

This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted:

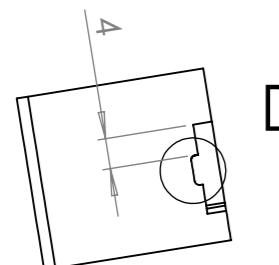
ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	18.05.2024 AMLL	18.05.2024 JLH	18.05.2024/TLS



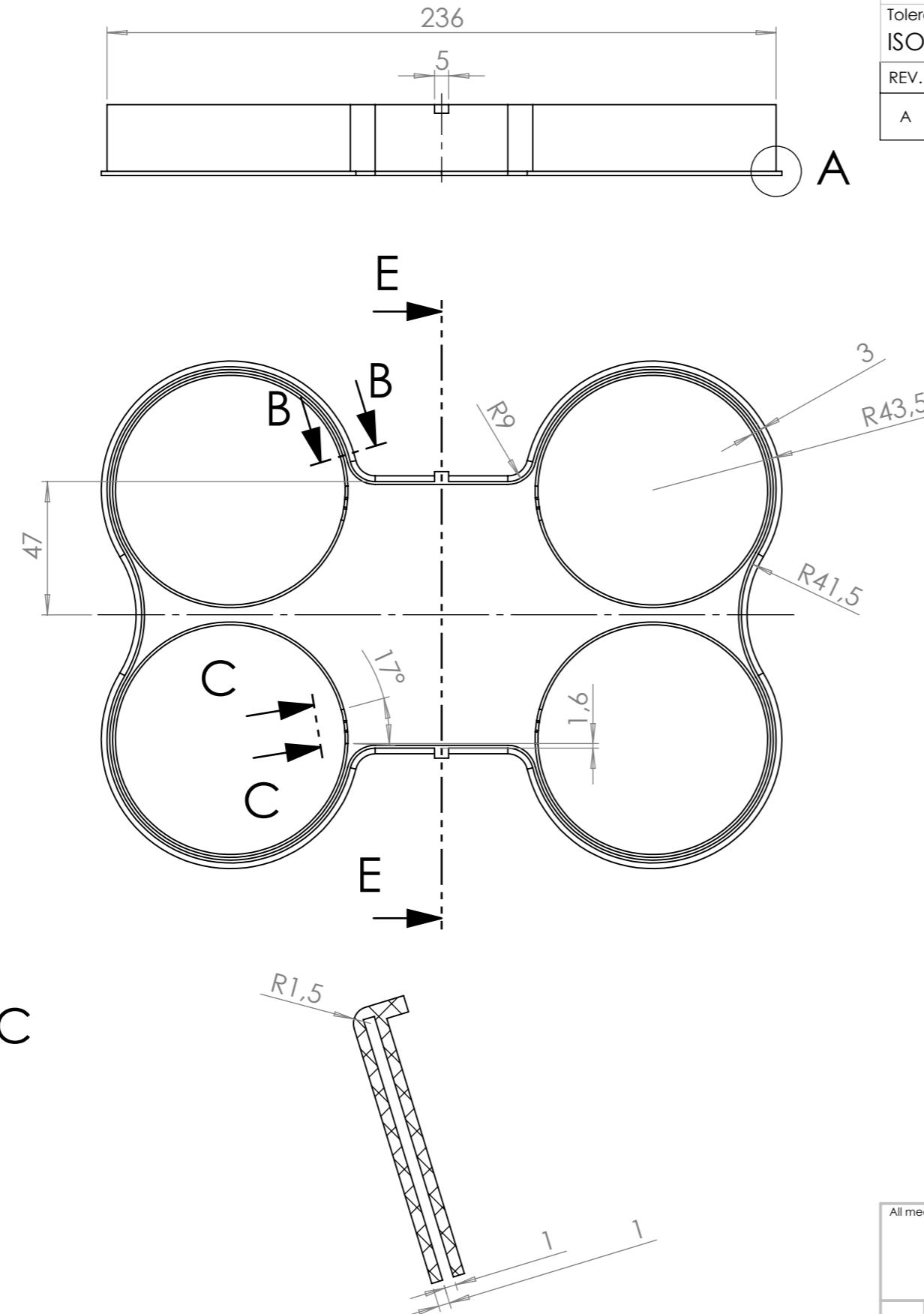
**DETAIL D**

SCALE 5:1  
Arm slot profile



**SECTION C-C**

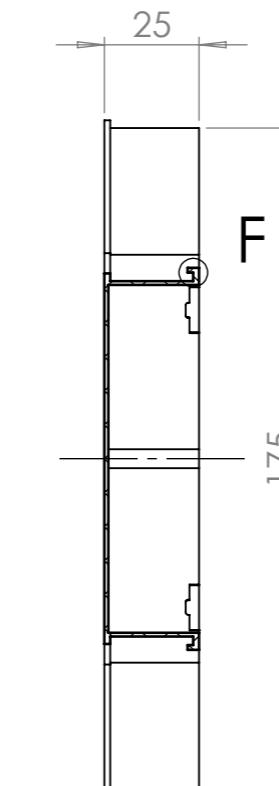
SCALE 1 : 1



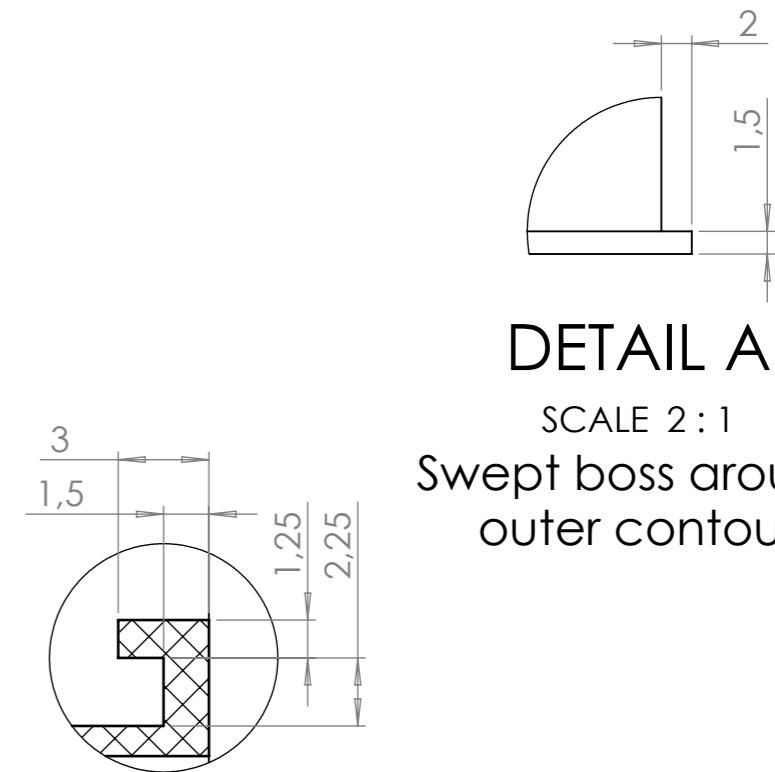
**SECTION B-B**

SCALE 2 : 1

Hollowed airframe with thin walls



**SECTION E-E**



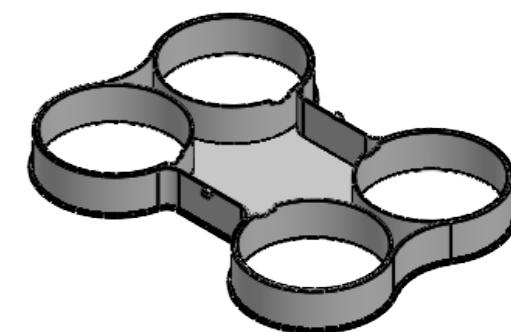
**DETAIL A**

SCALE 2 : 1

Swept boss around outer contour

**DETAIL F**

SCALE 4 : 1



All measurements are in mm U.N.O.				Projection:		
						DO NOT SCALE DRAWING
DRAWN	NAME	SIGNATURE	DATE	FINISH:	REVISION: A	
CHK'D				REPLACES:		
APP'D				REPLACED BY:		
Q.A.				Specification:	TITLE:	
				P-Filament 721	DWG NO.	
					A3	
				WEIGHT: 45.3g	SCALE: 1:2	
					SHEET 1 OF 1	

Airframe Top

drone-2

8

7

6

5

4

3

2

1

F

F

E

E

D

D

C

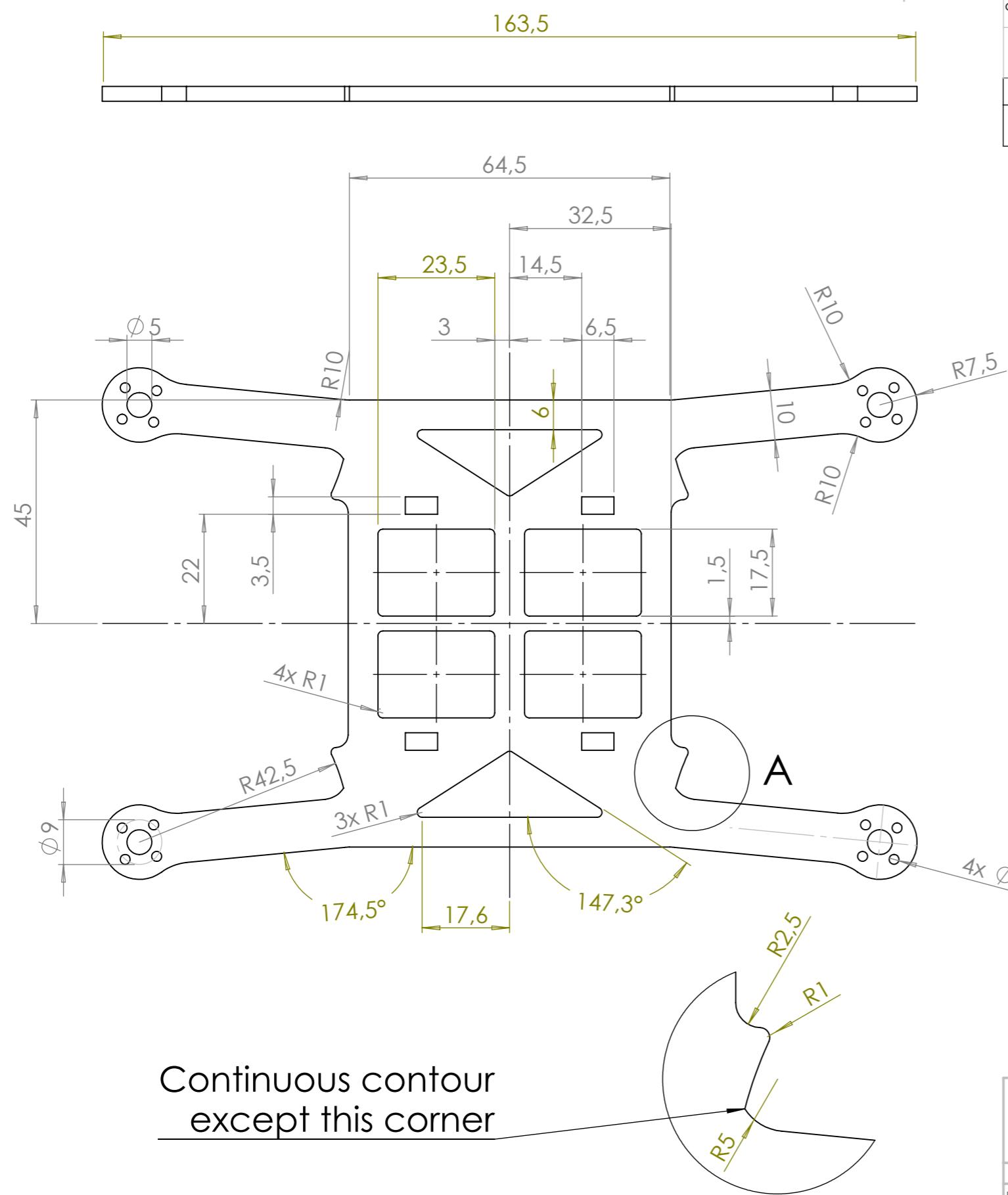
C

B

B

A

A

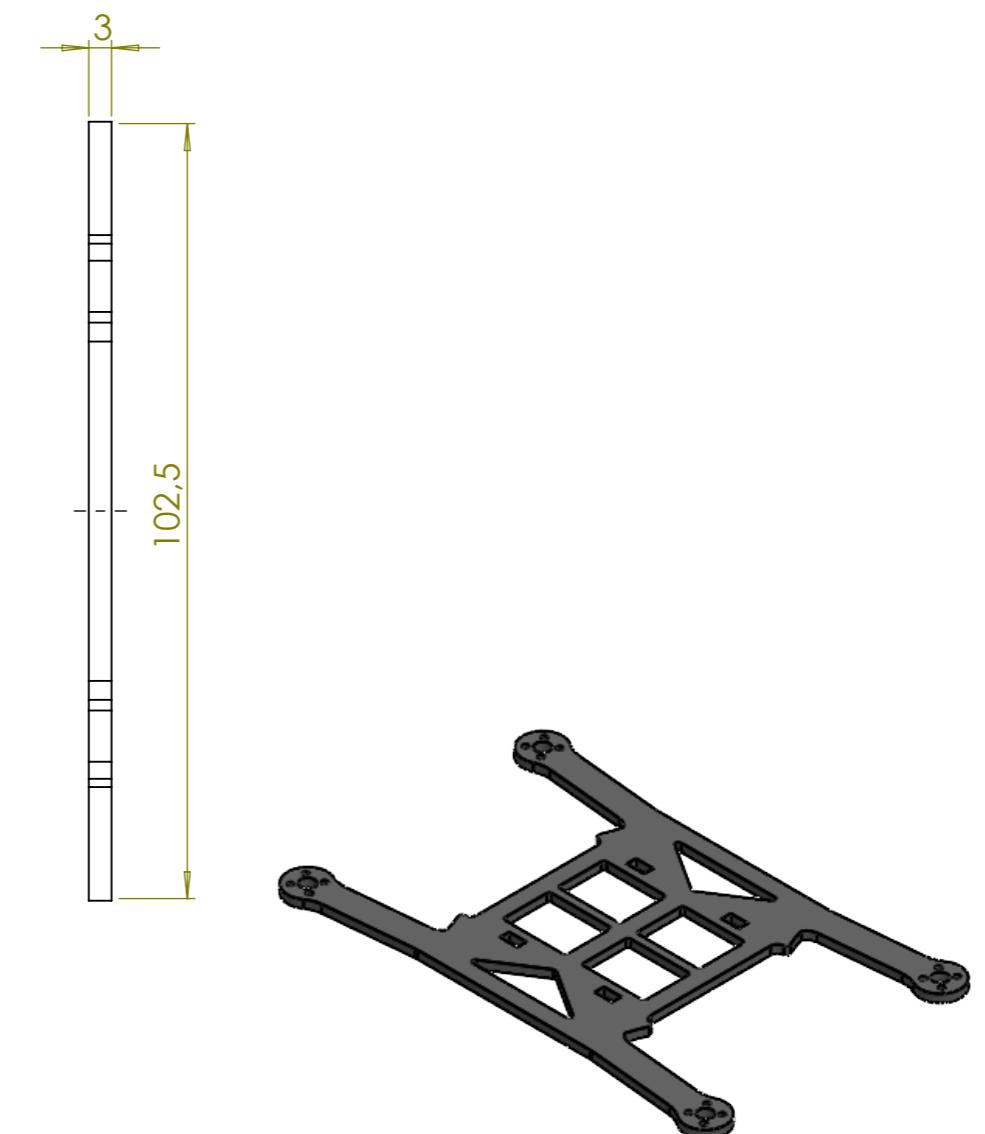


This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted:

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	18.05.2024 AMLL	18.05.2024 JLH	18.05.2024/TLS



All measurements are in mm U.N.O.				Projection:	DO NOT SCALE DRAWING		REVISION: A
DRAWN	NAME	SIGNATURE	DATE	FINISH:			
CHK'D				REPLACES:			
APP'D				REPLACED BY:			
Q.A.				Specification:		DWG NO.	
				PLA		drone-3	
				WEIGHT: 18.8g		SCALE:1:1	
						A3	
SHEET 1 OF 1							

University of Agder 

Airframe Core

8

7

6

5

4

3

2

1

F

F

E

E

D

D

C

C

B

B

A

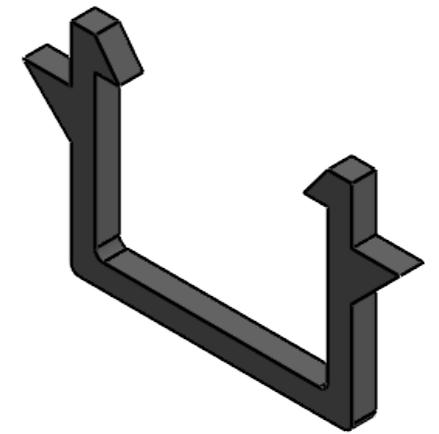
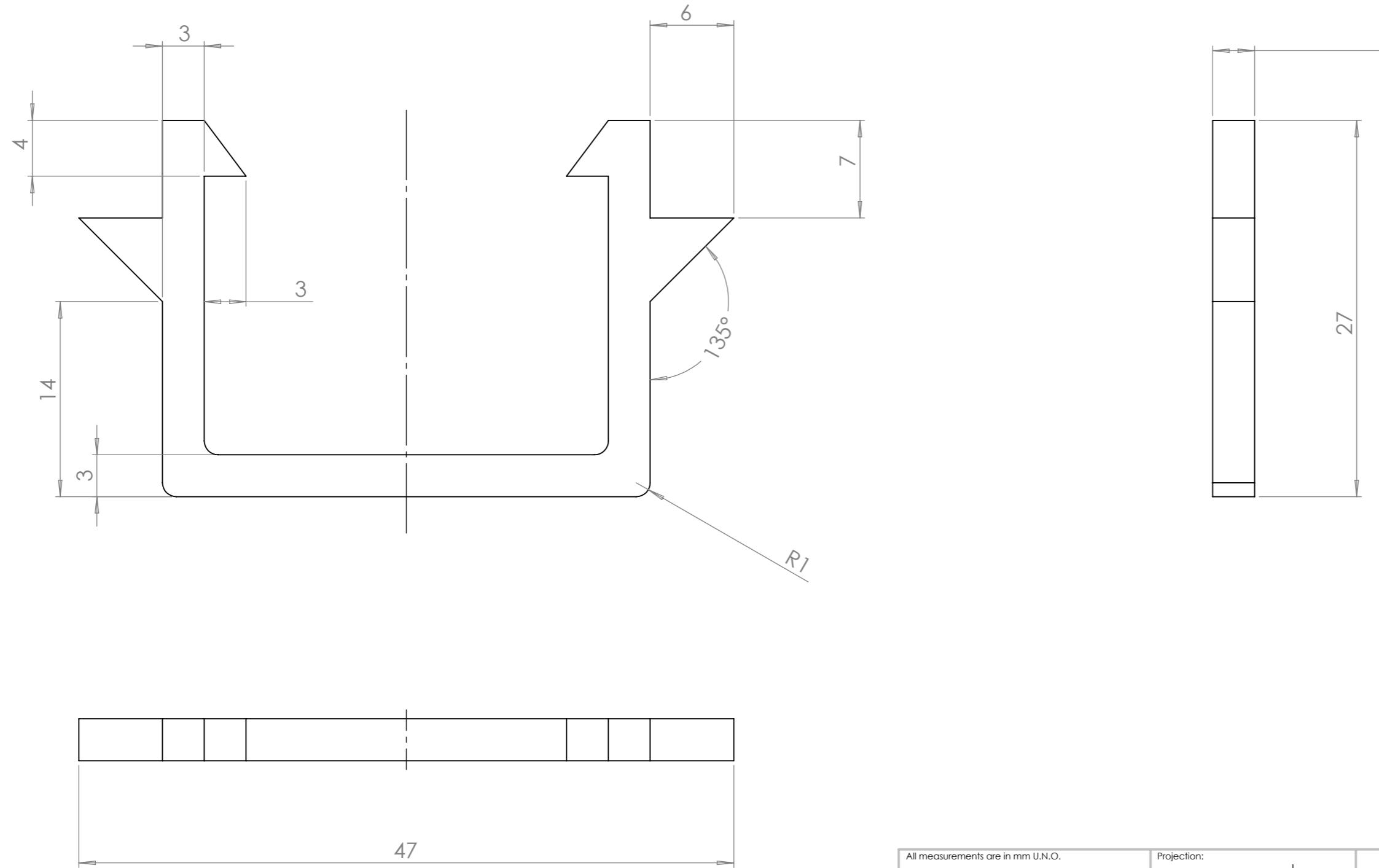
A

This document and all information and data herein or herewith are confidential and proprietary property of <COMPANY NAME> and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of <COMPANY NAME>

Tolerances except where noted:

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

REV.	DESCRIPTION OF REVISION	CHECKER	APPROVER	DATE/SIGN.
A	ISSUED FOR PRODUCTION	19.05.2024 AMLL	19.05.2024 JLH	19.05.2024/TLS



All measurements are in mm U.N.O.				Projection:	DO NOT SCALE DRAWING		REVISION: A
						University of Agder	
DRAWN	NAME	SIGNATURE	DATE	FINISH:	TITLE:		drone-4
CHK'D				REPLACES:			A3
APP'D				REPLACED BY:			
Q.A.				Specification:	DWG NO.		
				PLA			
				WEIGHT: 1.1g	SCALE: 3:1		
					SHEET 1 OF 1		

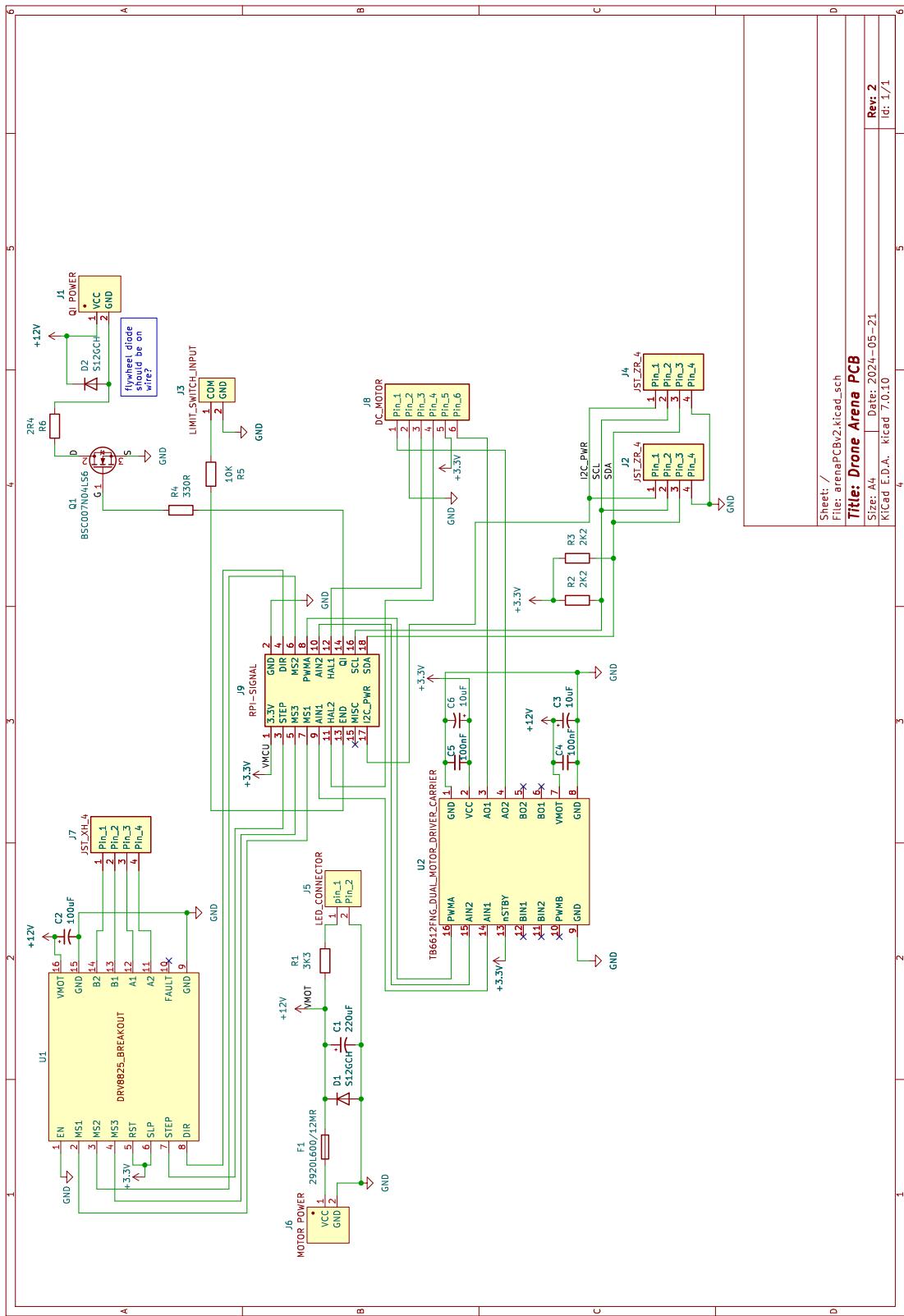
# B Bill of materials

## B.1 Bill of materials PCB

Part #	Component	Manufacturer	QTY	Description	PCB Symbol
1	2920L600/12MR	Littelfuse Inc.	1	PTC RESET FUSE 12V 6A	F1
2	B4B-ZR-SM4-TFT	JST Sales America Inc.	2	i2c connector	J2 & J4
3	B2B-ZR-SM4-TF	JST Sales America Inc.	1	limit switch connector	J3
4	TE Connectivity MTA-100 4 pin SMD 3-647166-4	TE Connectivity AMP Connectors	1	Stepper motor connector	J7
5	TE Connectivity MTA-100 2 pin SMD 3-647106-2	TE Connectivity AMP Connectors	1	QI lader connector	J1
6	Terminal Block 691508110302	Würth Elektronik	1	12V input	J6
7	S12GCH	Taiwan Semiconductor Corporation	2	Diode for QI charger and 12v input	D1 & D2
8	JST, ZH Female Connector Housing, 1.5mm Pitch, 4 Way, 1 Row	JST Sales America Inc.	2	i2c connector ZH fits in ZR	
9	JST, ZH Female Connector Housing, 1.5mm Pitch, 2 Way, 1 Row	JST Sales America Inc.	1		
10	JST ZH Series Female Crimp Terminal, 28AWG Min, 26AWG Max	JST Sales America Inc.	10	crimp for connector	
11	TE MTA-100 4 pin Female	TE Connectivity	1		
12	TE Connectivity MTA-100 Series Straight Surface Mount Pin Header, 6 Contact(s), 2.54mm Pitch, 1 Row (s), Unshrouded	TE Connectivity	1	DC motor connector	J8
13	TE MTA-100 6 pin Female	TE Connectivity	1		
14	TE MTA-100 2 pin Female	TE Connectivity	1		
15	Molex Milli-Grid Series Straight Surface Mount PCB Header, 18 Contact(s), 2.0mm Pitch, 2 Row(s), Shrouded	Molex	1	Rpi input	J9
16	Molex, Milli-Grid Female Connector, 18 Way, 2 Row	Molex	1		
17	Molex Milli-Grid Series Female Crimp Terminal	Molex	18		
18	2.54mm pitch Samtec TSM Right Angle Surface Mount Pin Header	Samtec	1	Pin header for LED	J5

19	Amphenol Communications Solutions Straight Surface Mount PCB Socket, 8-Contact, 1-Row, 2.54mm Pitch, Solder Termination	Amphenol Communications Solutions	4	Pin holes for drivers	U1 & U2
20	BSC007N04LS6	Infeon	1	Mosfet	Q1
21	1210 KYOCERA AVX 100nF	KYOCERA AVX	2	100nF ceramic	C4 & C5
22	CHEMI-CON 10µF Aluminium Electrolytic Capacitor 50V dc, Surface Mount	CHEMI-CON	2	10uF	C3 & C6
23	NIC Components 220µF Aluminium Electrolytic Capacitor 50V dc	NIC Components	1	220uF	C1
24	KEMET 100µF Aluminium Electrolytic Capacitor 50V dc	KEMET	1	100uF	C2
25	E Connectivity 3.3kΩ, 2512, 3W	TE Connectivity	1	3K3, min 2W	R1
26	ROHM 2.2kΩ, 1210	ROHM	2	2K2	R2 & R3
27	330R 1210 SMD	Panasonic	1	330R	R4

# C Arena PCB Schematic



## D Project Management

## D.1 Gantt Chart

As the chart is too long to display in one long row, it was split in three. This section includes a snapshot of the Gantt chart in the earlier stages of the project, so the content is not up to date, but it demonstrates the planning process.

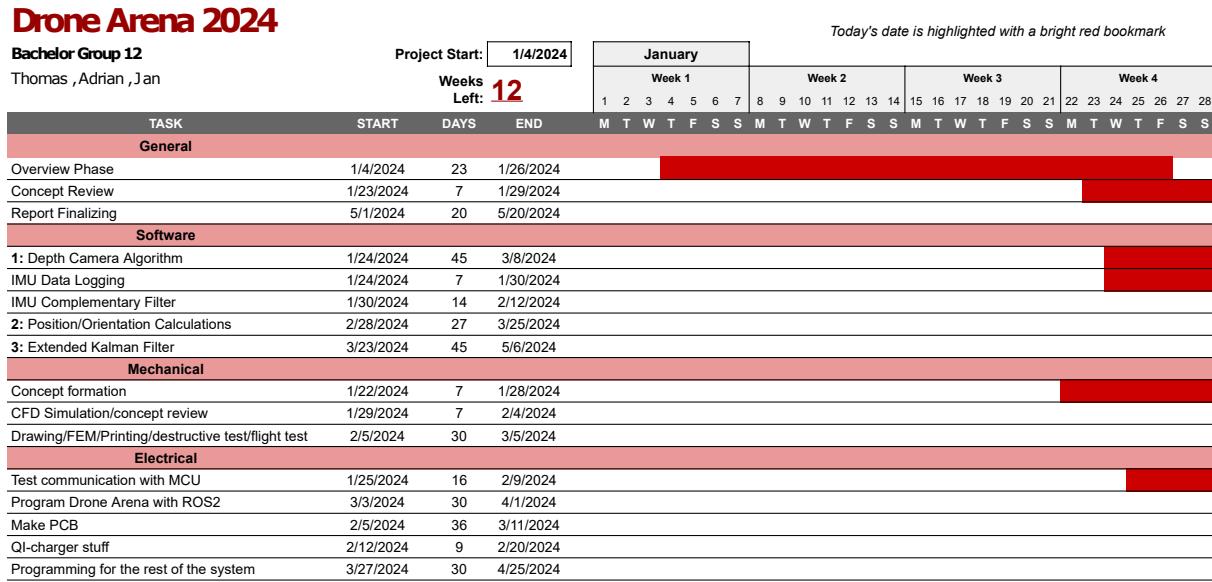


Figure D.1: Gantt chart created in Google sheets: part 1

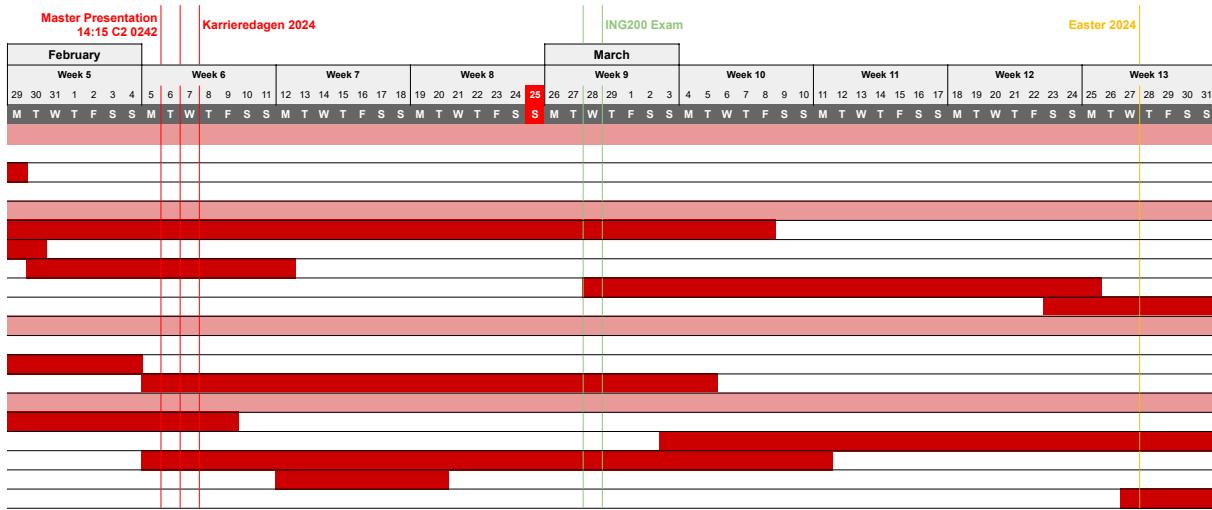


Figure D.2: Gantt chart created in Google sheets: part 2

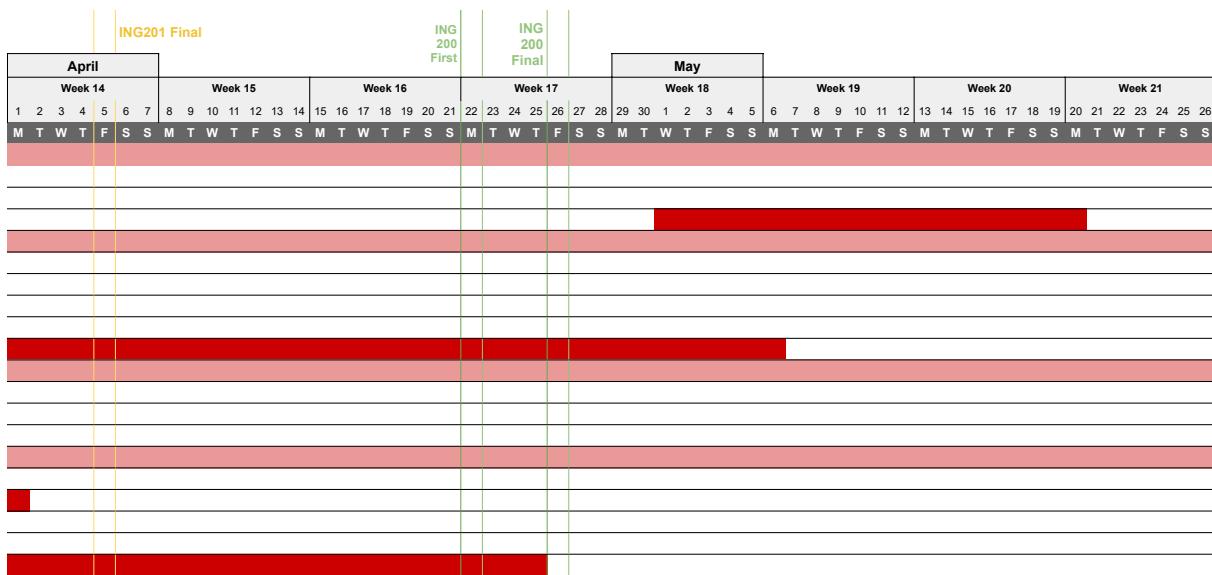


Figure D.3: Gantt chart created in Google sheets: part 3

## D.2 Kanban Board

The Kanban board was updated throughout the bachelor project, but a image was taken during development to display the progress, shown in figure D.4.

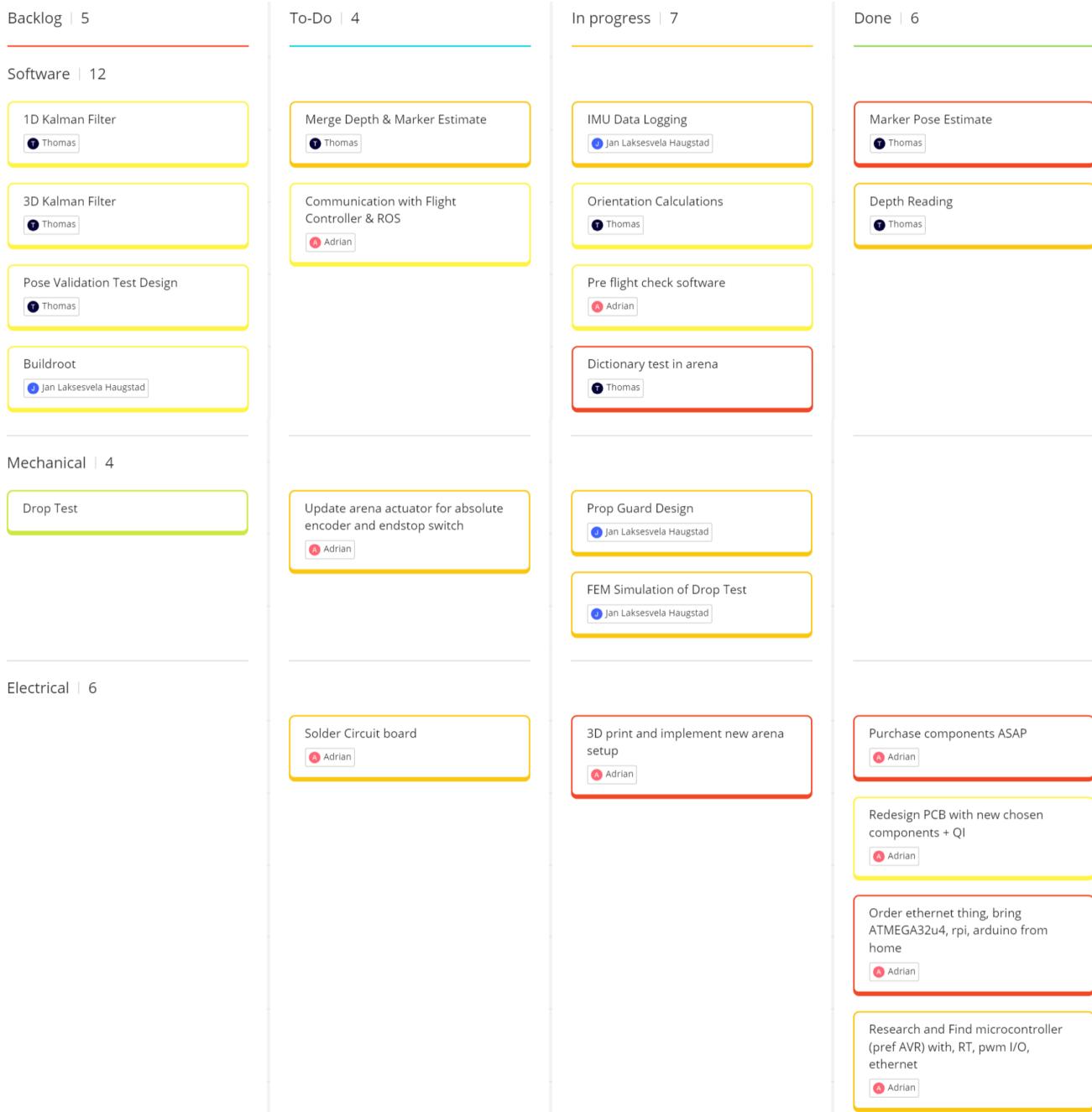


Figure D.4: Kanban board created in Miro.

# E Pose Validation Test

## E.1 Test Plan

*Test Plan*

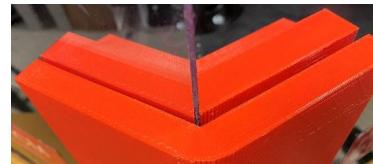


## POSE VALIDATION OF VISION SYSTEM WITH MOTION CAPTURE

This test serves the purpose of validating the performance of the machine vision system in the drone reinforcement learning arena. It utilizes the Qualisys motion capture equipment at UiA, which has a high accuracy and can be used for comparison since the orientation and position is simple to measure from it. The test is tailored to small markers on the drone, with the cube's surface being  $52 \times 52$  millimeters. In addition to logging the markers pose, it also allows for further analysis (such as error distribution) which is desired for state estimator filters. The test is designed with physical calibration tools and scripts from "Code/QualisysTest" in the repository made for the corresponding bachelor project.

### Procedure:

1. Identify the working volume of the system. Take off every other transparent wall plate, so the arena is standing on three plates. This gives space for the test and keeps the camera mount intact. Also take out the triangular metal plates on the floor of the arena.
2. Place the calibration plate with the large marker and Qualisys calibration tool on top of the middle actuator. Move the actuator down until it is physically stopped by the 3D printed bracket. Align the three transparent walls edges with the inner corner of the red print as shown in the figure to the right. Use something to keep them from gliding in the slit (duct tape on both sides works well).

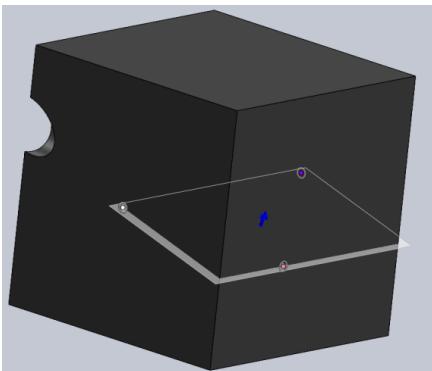


## Test Plan

3. Run the “poseTestInitializer.py” script for the calibration sequence. Rotate the plate until the marker is aligned with the camera (which also is parallel to the transparent wall plate). The script prints a string with information of when the axes are within the specified tolerance. Press Q on the keyboard while in the display window to exit after the plate is inside the tolerance.
4. Measure and write down the three physical distances for the translation between the Qualisys frame and the camera frame. To visualize the center of the camera, utilize the crosshair display in the script “cameraCenter.py”. The center of the Qualisys calibration tool is in the bottom of the outer corner of the L-frame. Compare the translation vector to this (which was measured in mm during development):  $[ \begin{array}{ccc} x & y & z \end{array} ] = [ \begin{array}{ccc} 190 & 143 & 1564 \end{array} ]$
5. Set up the Qualisys motion capture cameras and orient them so at least 4 can see the full L-frame to define the global reference frame. Furthermore, adjust the exposure and flash time (during development  $20 \mu\text{s}$  were sufficient for most cameras) and set up masks to ignore false positives, such as reflections or external light sources. The motion capture cameras may impact each other, which can be fixed by introducing a small phase shift by placing them in different exposure delay groups.
6. Calibrate the working volume with the Qualisys calibration tool and the calibration wand. After the calibration is complete, place a piece of paper over the large ArUco marker so it does not interfere with the machine vision tracking.

## *Test Plan*

7. Create a new 6DOF rigid body for the cube, so this may be set up as the 6DOF calculations under the “Reprocess” setting after the test is recorded to export the movement of the cube and not just the reflective markers. Make sure to translate and rotate the local frame for this body so it is placed in the middle of the ArUco marker, oriented the same way as in OpenCV. (Tip: use a reference plane and the angle relationship between the ArUco plane in a CAD software to orient it aligned with the ArUco plane. The angle between the ArUco plane and reference plane on the right was 16.4 degrees, where the latter plane was defined by the three center points selected in the figure)
8. Stick the marker on the motion capture cube with double sided tape (so it can be replaced later for new tests). A slim white edge may be printed outside the marker to separate it from the cube.
9. Validate the Qualisys coordinates by moving the cube to along each axis and reading the coordinates of the 6DOF rigid body. This can be done by entering “View>Data Info 1”, then right clicking the black pane and selecting “Display 6DOF data” in QTM.
10. Start the recordings with both the Qualisys pose and the “poseTestRecording.py” script. The data from Qualisys and frames from the D435 camera recording may be synchronized through a common movement, for example keeping the cube still from the beginning of the test and registering the first change in position as the start time. This method was utilized in the script



## Test Plan

“poseTestSynchExport.py”, which can be used as inspiration for later experiments. After the initial movement, move the marker around in the arena in multiple locations and orientations for a dynamic display. Multiple recordings may be desired to gain sufficient data (recommended at least 2-3). Stop the “poseTestRecording.py” script by pressing Q on the keyboard while in the display window.

11. As the filenames are automatically incremented when recording multiple tests, these can be run back-to-back without changing the name in the script. After the recording, a csv file with the frame and timestamp, as well as a raw video will be saved. For all other files, remember to change the variable “testNr” to correspond with the number of the recorded raw data to be exported and analyzed. Marker detection and pose estimation from this data is done in another script, with this project’s algorithm shown in “poseTestArucoExport.py”. This exports a video with the axes displayed and more data. Finally, the “poseTestSynchExport.py” script synchronizes the data and transforms them to the same reference frame, before exporting them to a single csv file.

### Success Criteria:

1. Logged data from both motion capture and marker system.
2. Same number of frames, so the data can be correctly analyzed.
3. Same start movement detected. Verify by manually inspecting the recordings, that the movement and timestamps are correct.
4. No incorrect poses from motion capture system. This may happen if not enough markers are detected to estimate a rotationally unique pose. Such situations may also arise if the cameras’ views are obscured.

## E.2 File Overview

Table E.1: Overview of files and contents for pose validation test

poseTestRecording.py		Qualisys Track Manager	
Contents	Description	Contents	Description
Frame, Timestamp	Raw data from the D435 recording. Internal timestamp from when D435 streaming began		Raw data from QTM, internal timestamp from when motion capture began. Rotation matrices elements are given in this order, with naming convention Rot[0] to Rot[8]
poseTestArucoExport.py			
Contents	Description	Contents	Description
Frame, Timestamp, tVec0_0, tVec0_1, tVec0_2, tVec1_0, tVec1_1, tVec1_2, rVec0_0, rVec0_1, rVec0_2, rVec1_0, rVec1_1, rVec1_2, reprojError0, reprojError1	Same frames and timestamp as before, but rows without detected marker have been removed. Exports translation vector, rotation vector and reprojection error for each solution per frame	Frame Number, Timestamp, X,Y,Z, Roll, Pitch, Yaw, Residual, R11, R12, R13, R21, R22, R23, R31, R32, R33	Recorded Video with displayed pose estimation through fiducial marker axes
poseTestSyncExport.py			
Contents	Description		
Time, xQTM, yQTM, zQTM, xCV0, yCV0, zCV0, v0R11, v0R12, v0R13, v0R21, v0R22, v0R23, v0R31, v0R32, v0R33, v1R11, v1R12, v1R13, v1R21, v1R22, v1R23, v1R31, v1R32, v1R33, qtmR11, qtmR12, qtmR13, qtmR21, qtmR22, qtmR23, qtmR31, qtmR32, qtmR33, xCV1, yCV1, zCV1	"Synchronized" timestamp, translation and rotation from QTM as well as the rotation matrices from Rodrigues() method on each solution of the rotation vectors		

### E.3 Pose Validation Test Results: Plots from Full Time Range

#### E.3.1 Test #0 Results

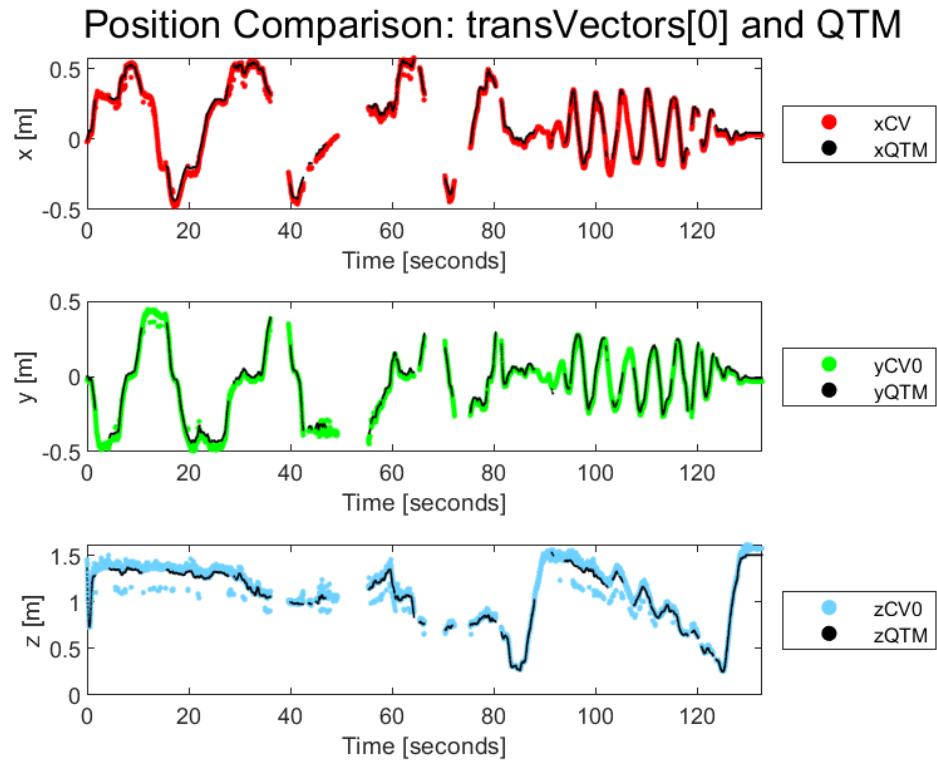


Figure E.1: Position plot, comparing elements of translation vectors

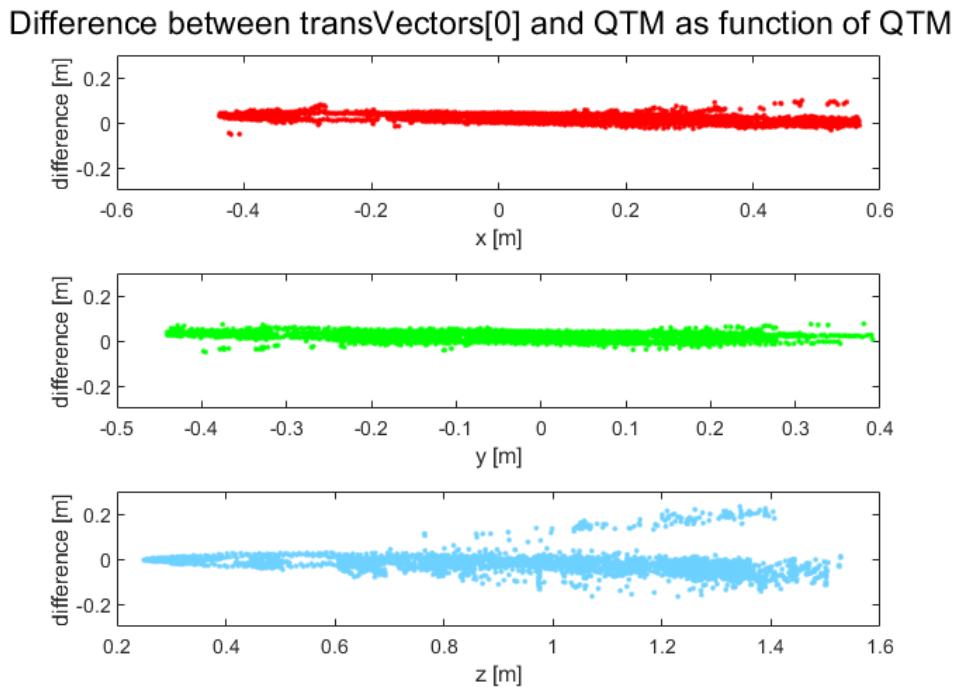


Figure E.2: Difference between first estimated translation and QTM, as function of QTM position

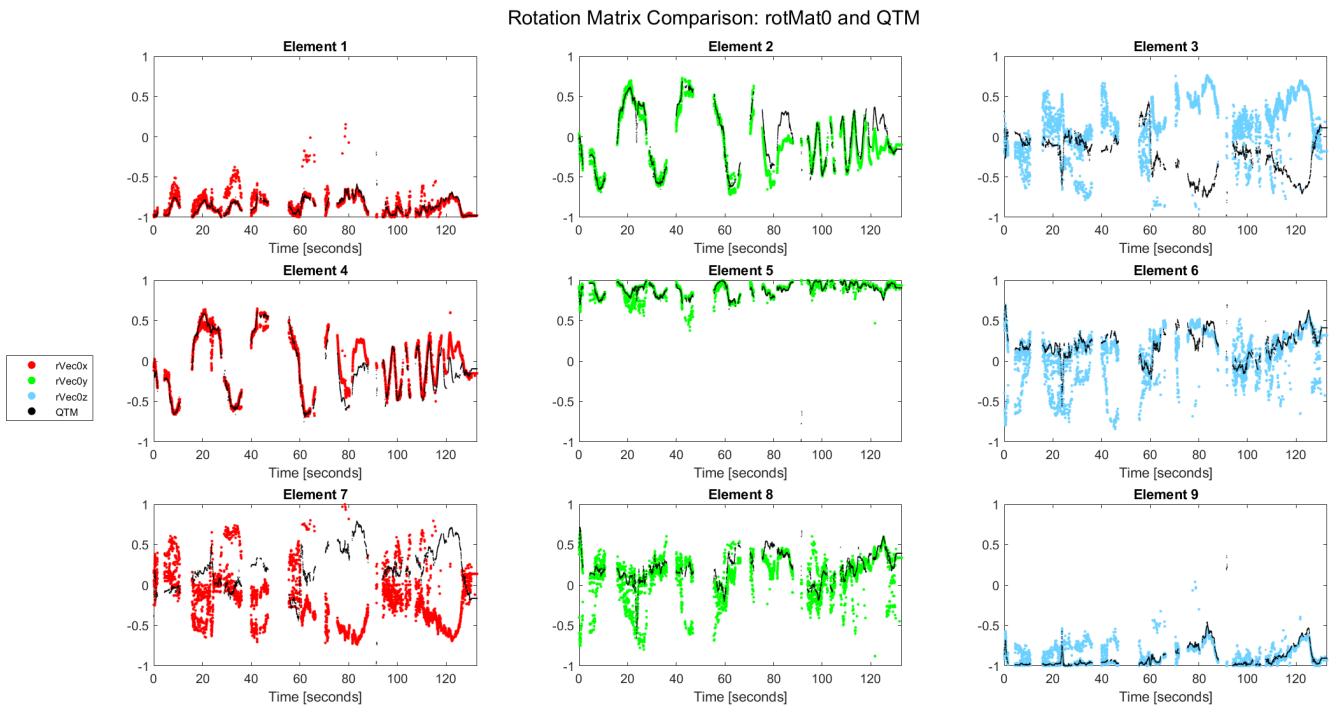


Figure E.3: Comparison of QTM and first estimate's rotation matrix elements

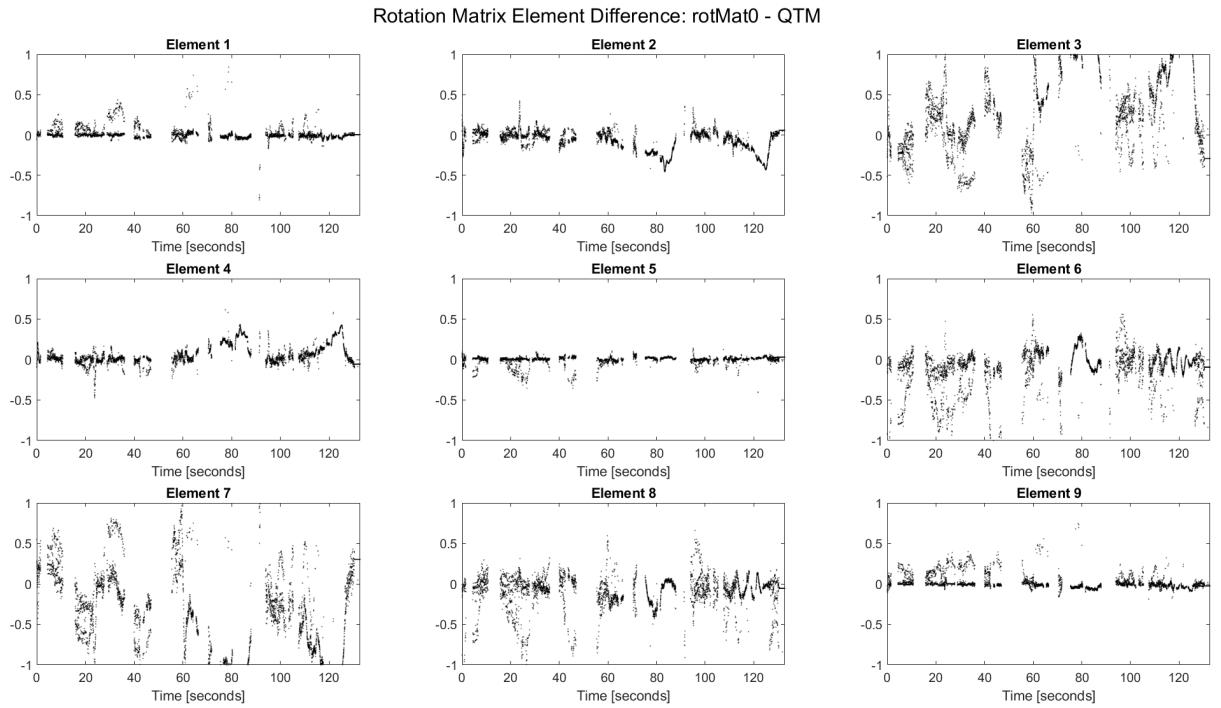


Figure E.4: Difference between QTM and first estimate's rotation matrix elements

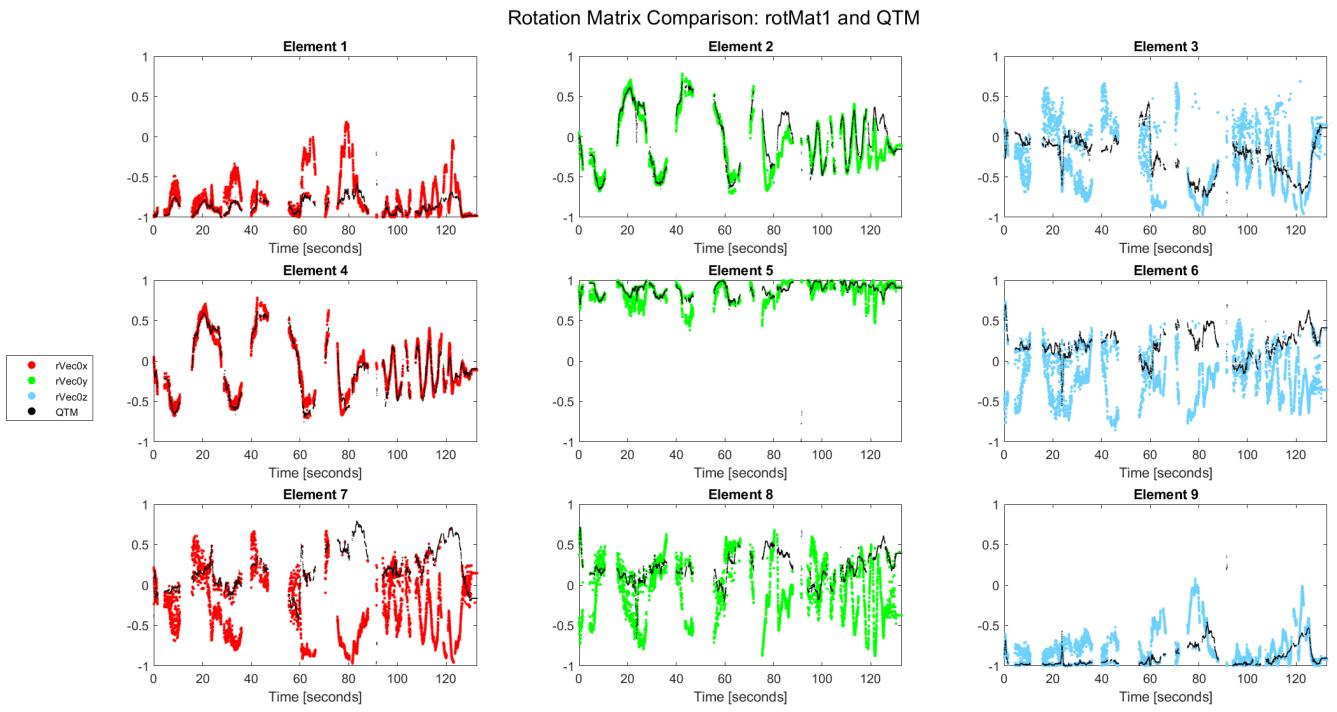


Figure E.5: Comparison of QTM and second estimate's rotation matrix elements

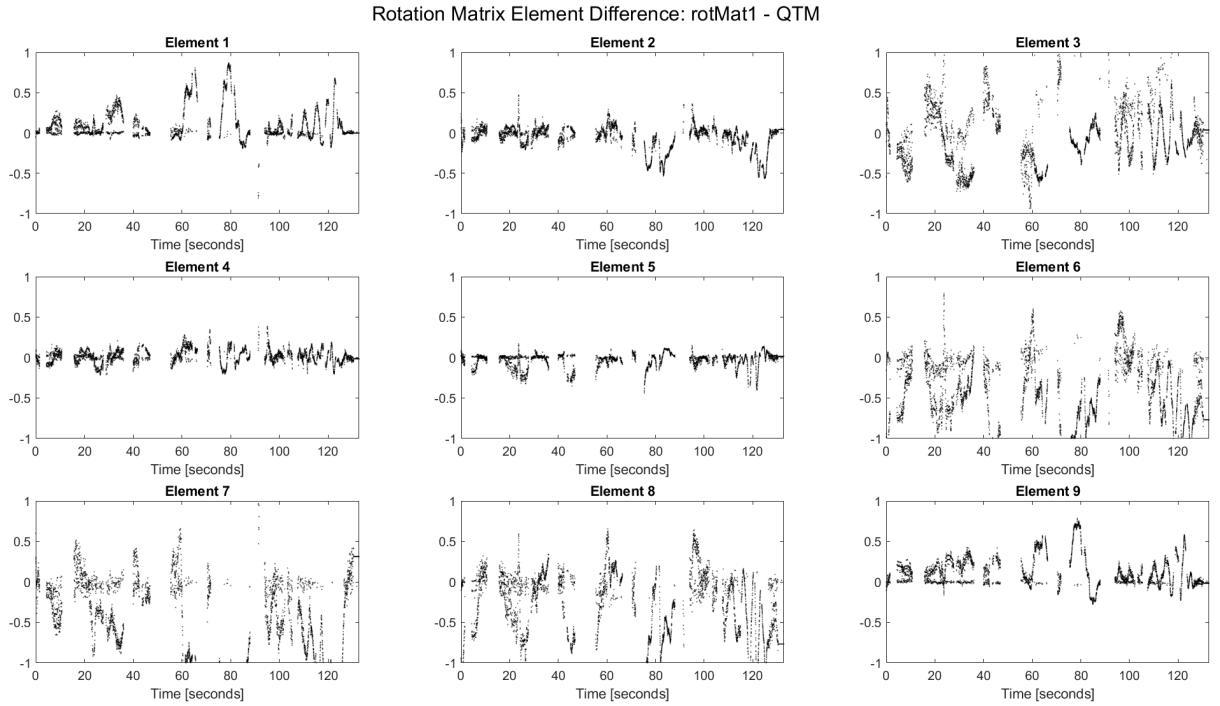


Figure E.6: Difference between QTM and second estimate's rotation matrix elements

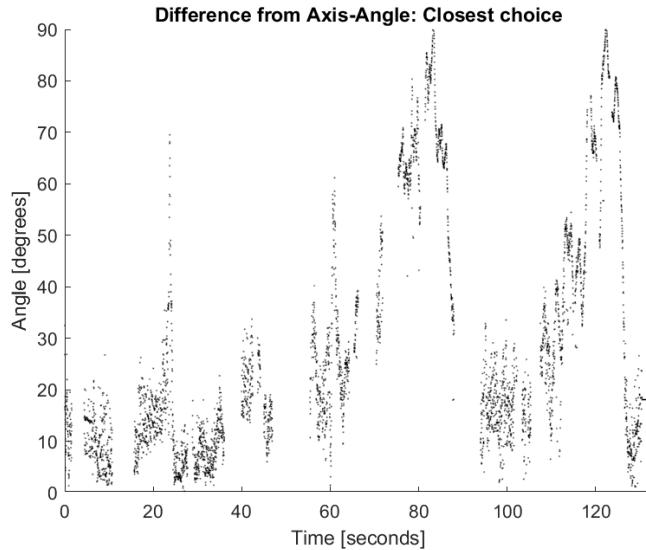


Figure E.7: Axis-Angle plot of angular difference between QTM and closest estimate

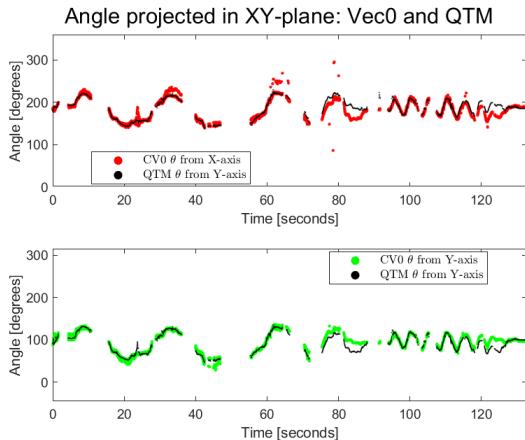


Figure E.8: XY-projected angle between global and first estimated axes, compared to QTM

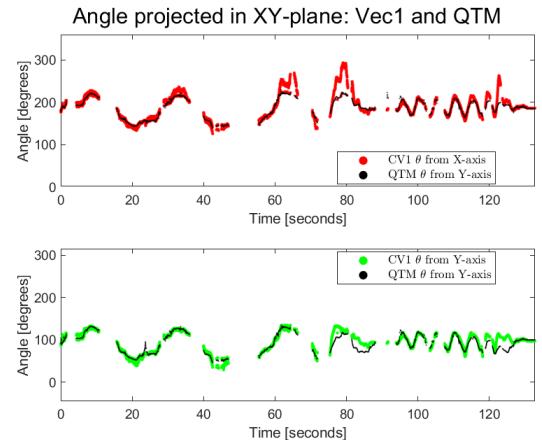


Figure E.9: XY-projected angle between global and second estimated axes, compared to QTM

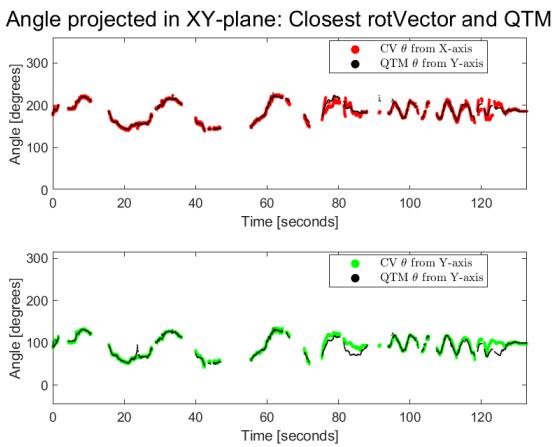


Figure E.10: XY-projected angle between global and chosen estimated axes, compared to QTM

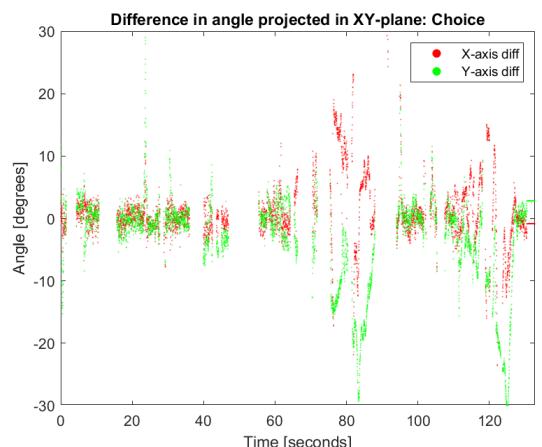


Figure E.11: XY-projected angle difference between chosen estimate and QTM axes

### E.3.2 Test #1 Results

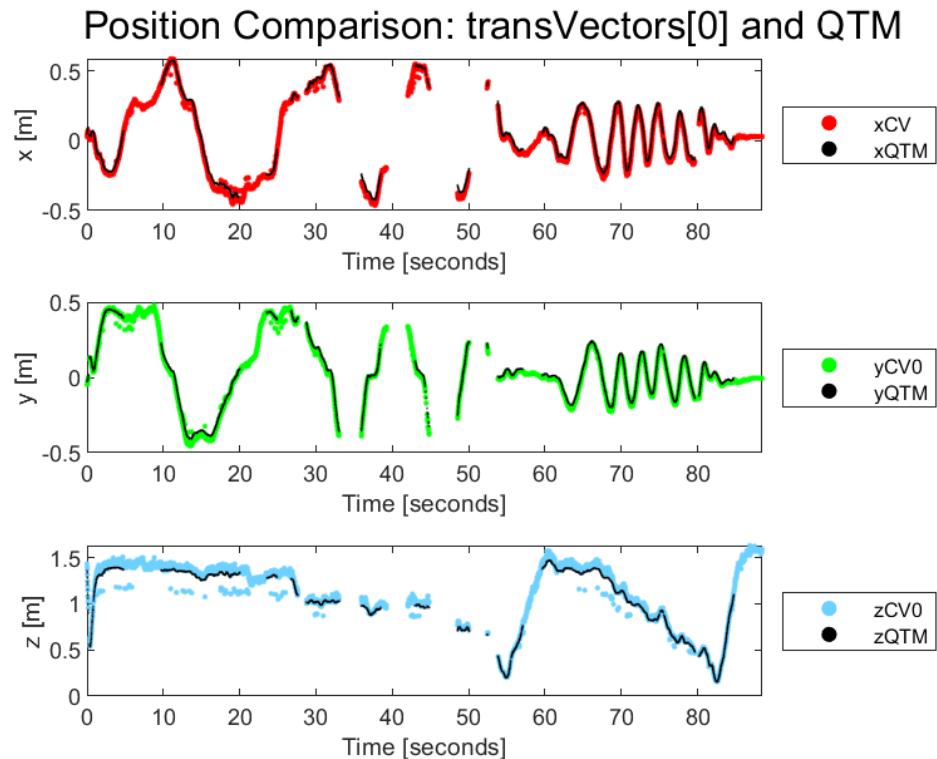


Figure E.12: Position plot, comparing elements of translation vectors

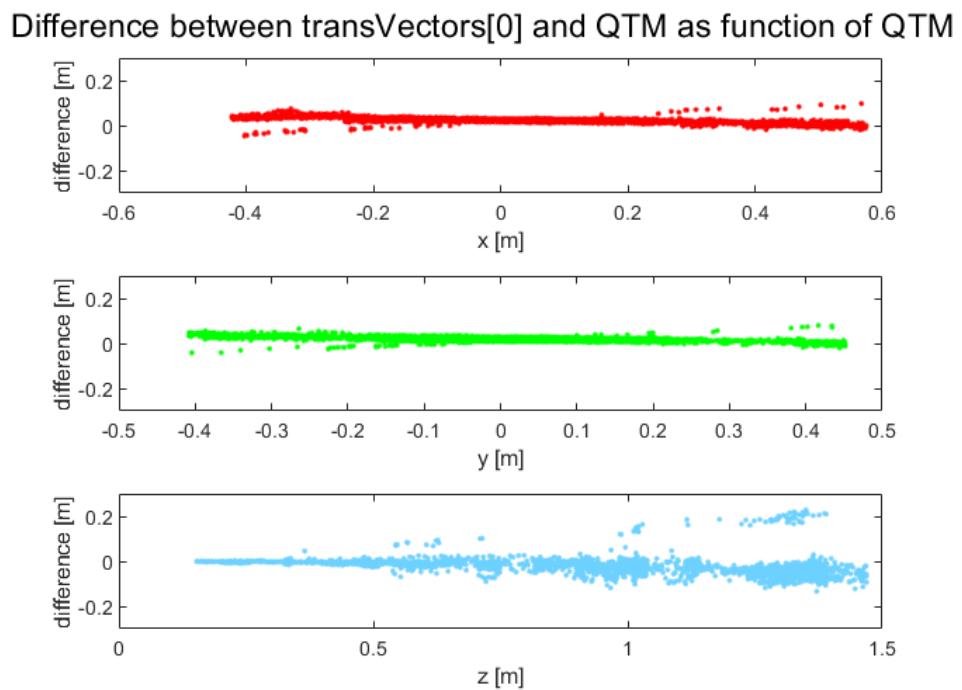


Figure E.13: Difference between first estimated translation and QTM, as function of QTM position

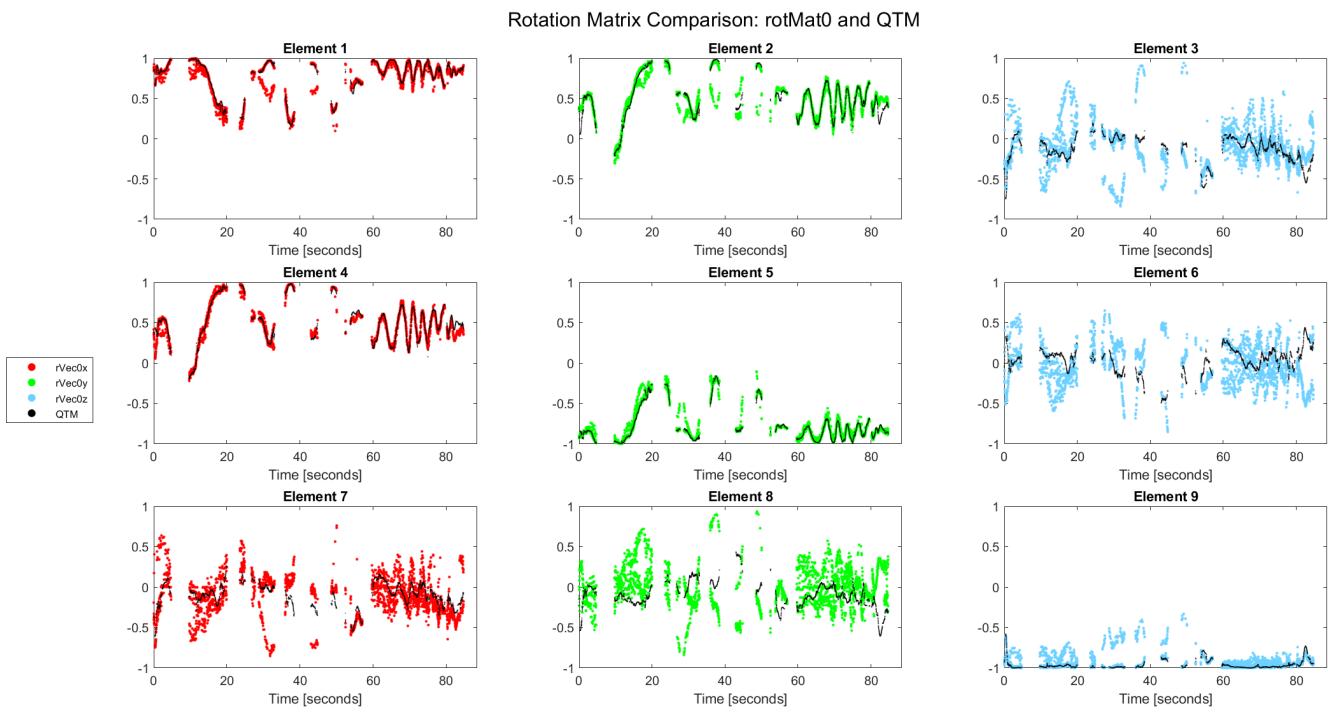


Figure E.14: Comparison of QTM and first estimate's rotation matrix elements

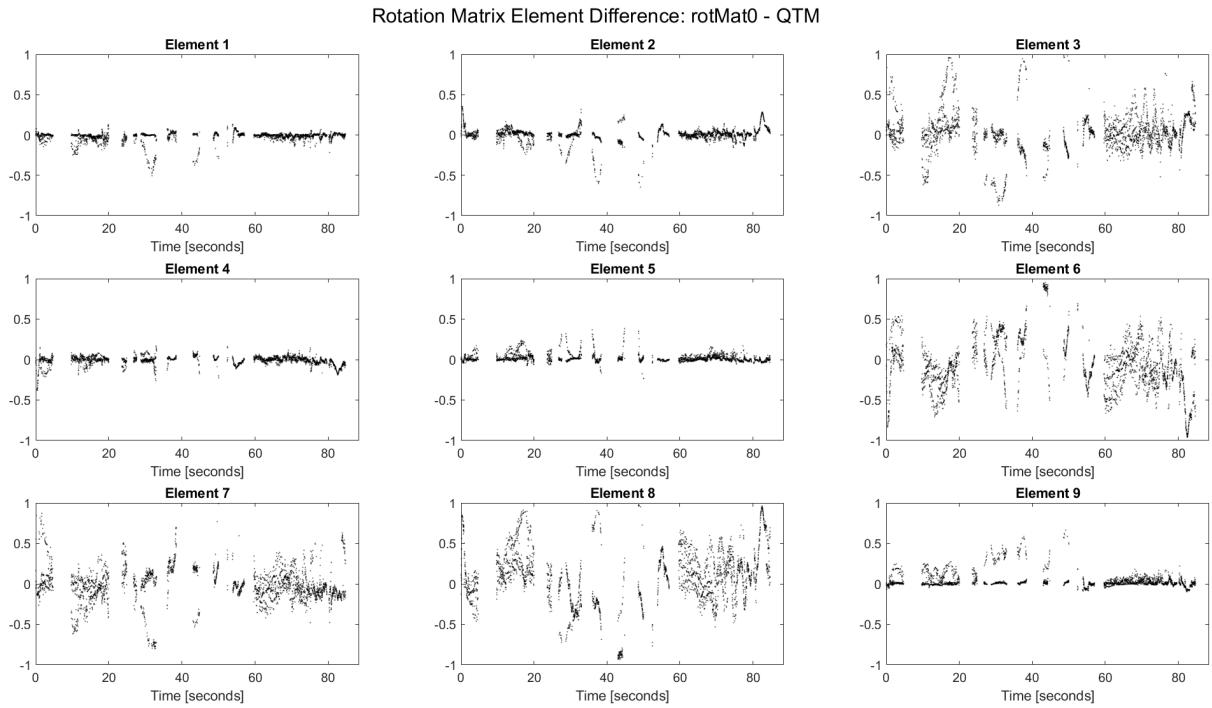


Figure E.15: Difference between QTM and first estimate's rotation matrix elements

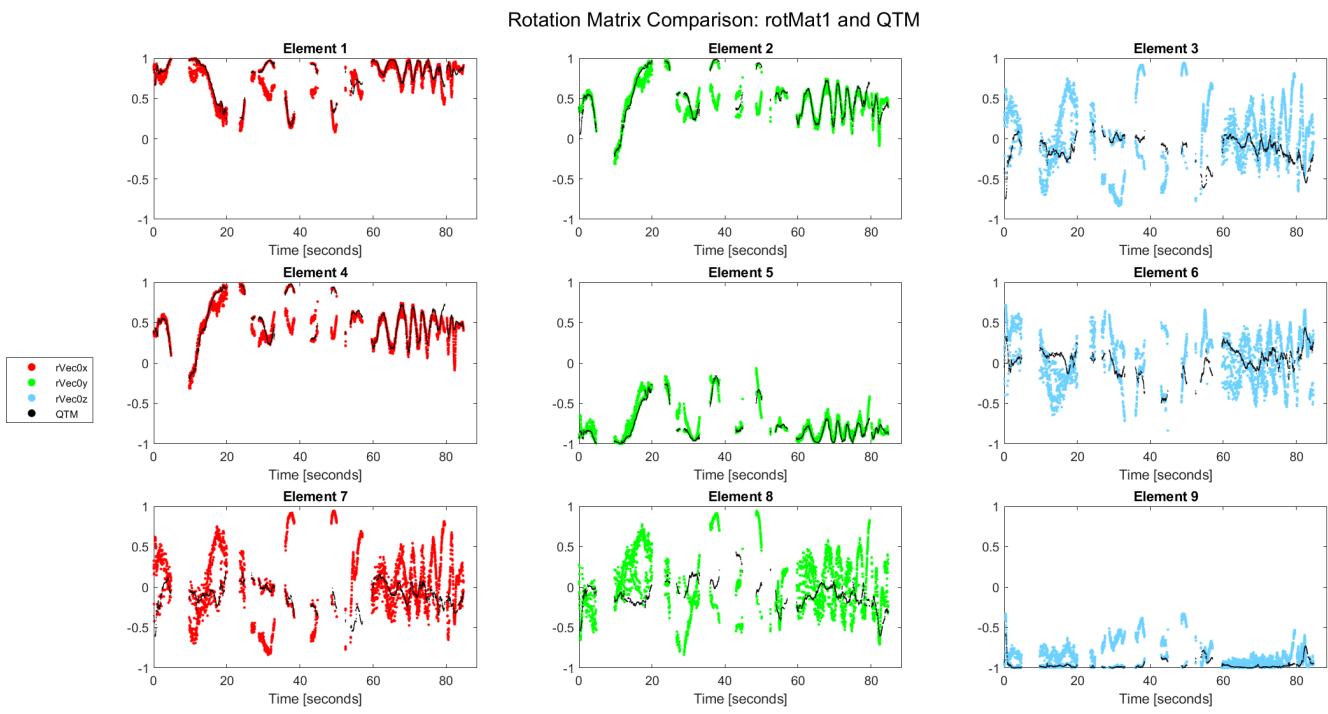


Figure E.16: Comparison of QTM and second estimate's rotation matrix elements

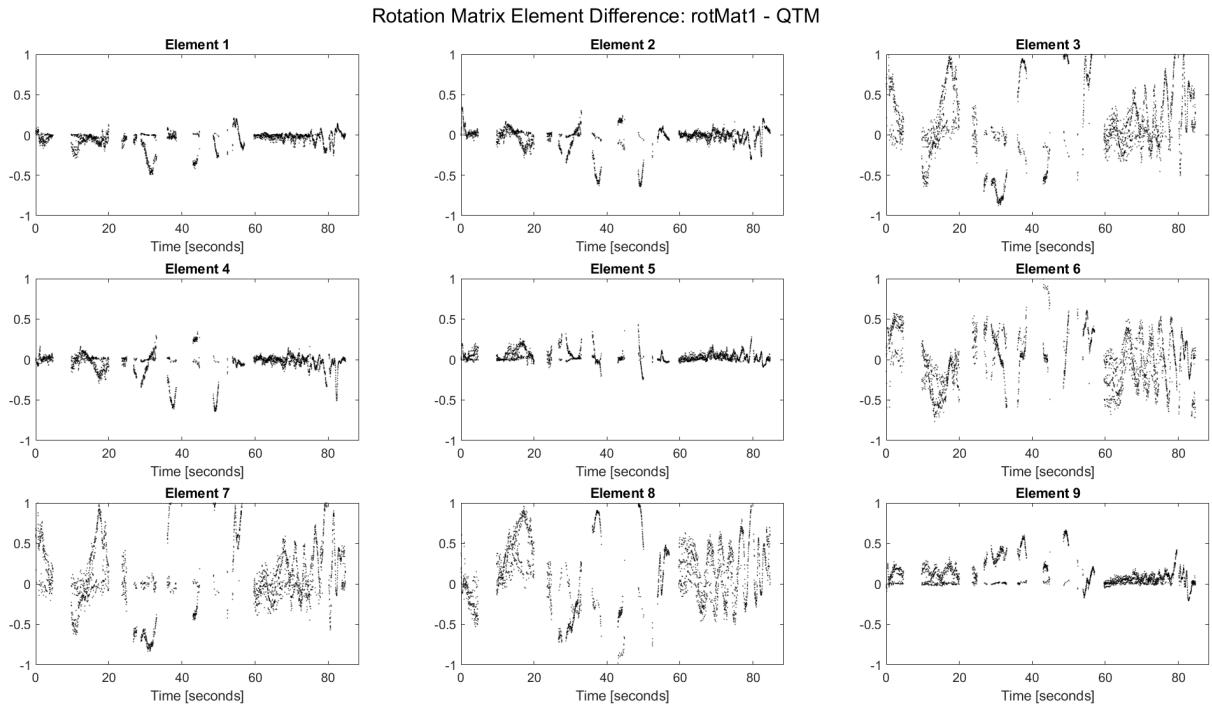


Figure E.17: Difference between QTM and second estimate's rotation matrix elements

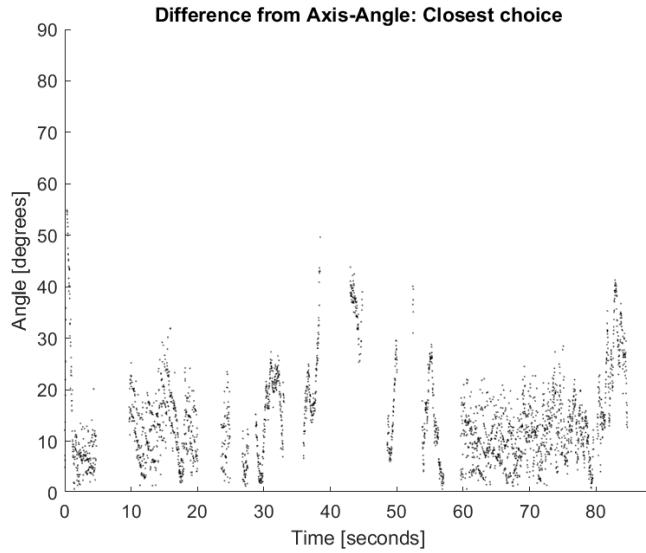


Figure E.18: Axis-Angle plot of angular difference between QTM and closest estimate

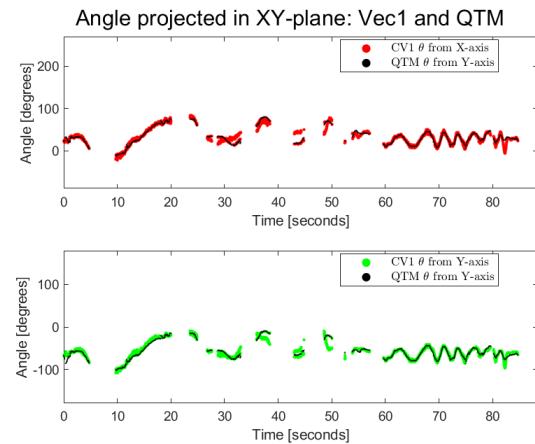
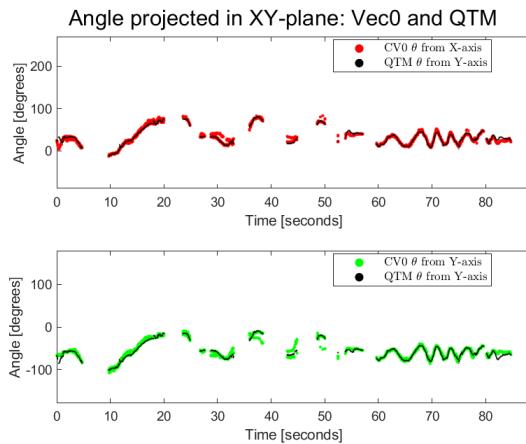


Figure E.19: XY-projected angle between global and first estimated axes, compared to QTM

Figure E.20: XY-projected angle between global and second estimated axes, compared to QTM

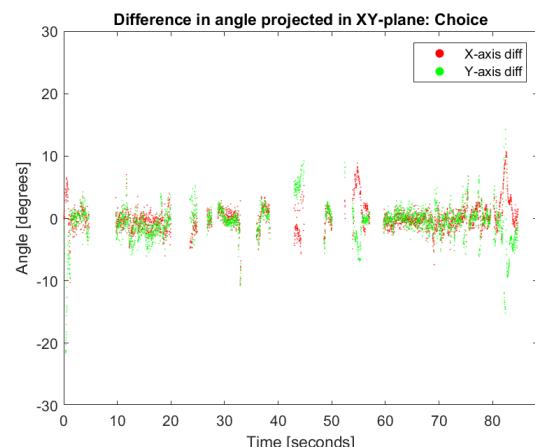
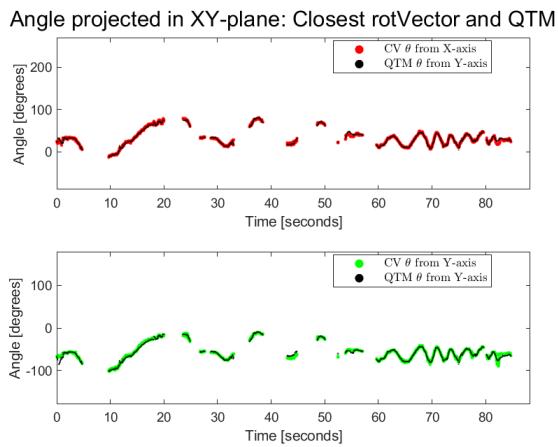


Figure E.21: XY-projected angle between global and chosen estimated axes, compared to QTM

Figure E.22: XY-projected angle difference between chosen estimate and QTM axes

### E.3.3 Test #2 Results

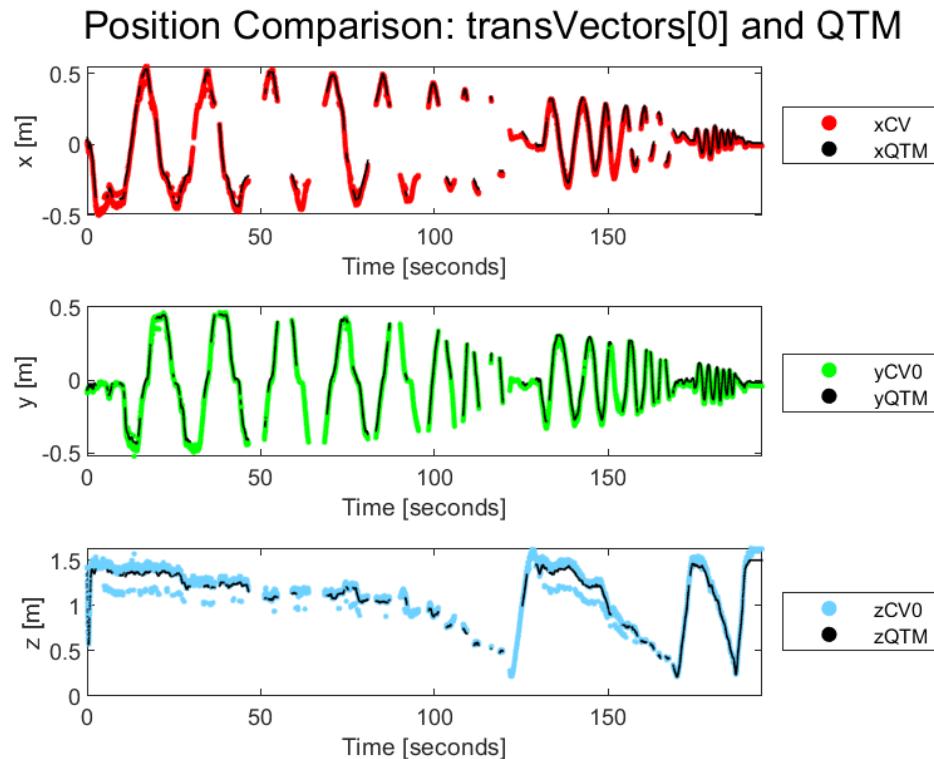


Figure E.23: Position plot, comparing elements of translation vectors

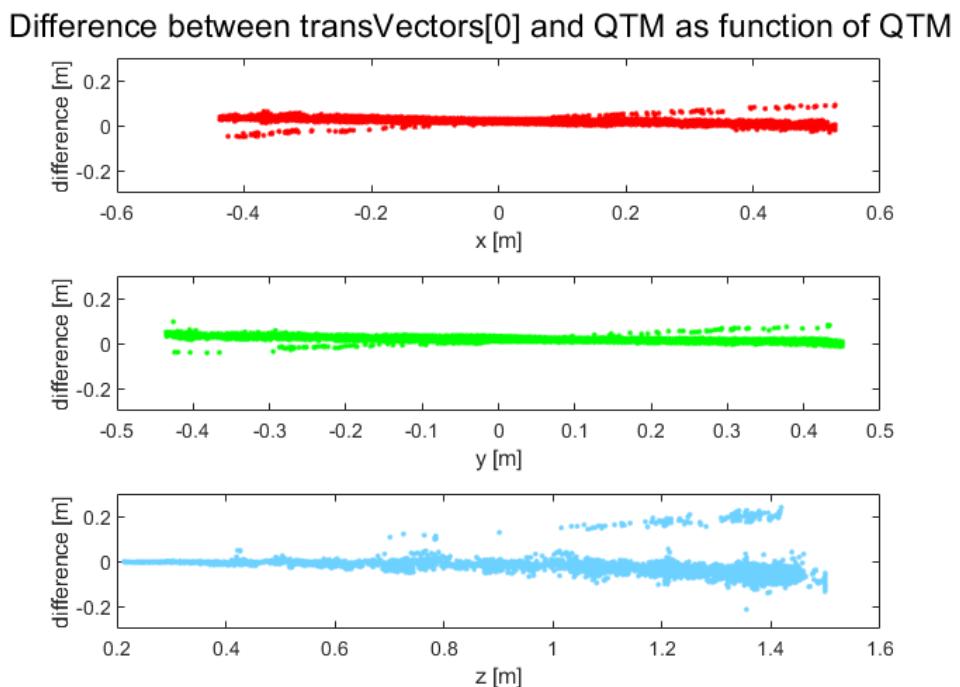


Figure E.24: Difference between first estimated translation and QTM, as function of QTM position

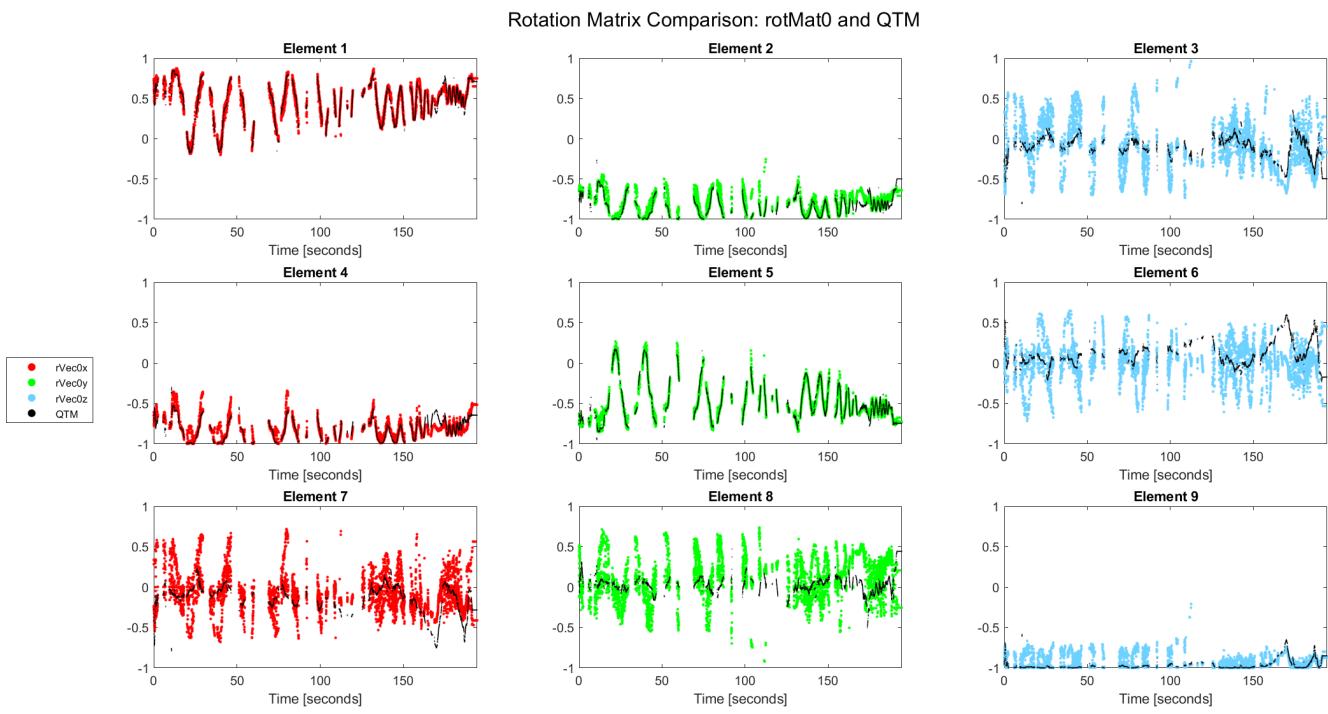


Figure E.25: Comparison of QTM and first estimate's rotation matrix elements

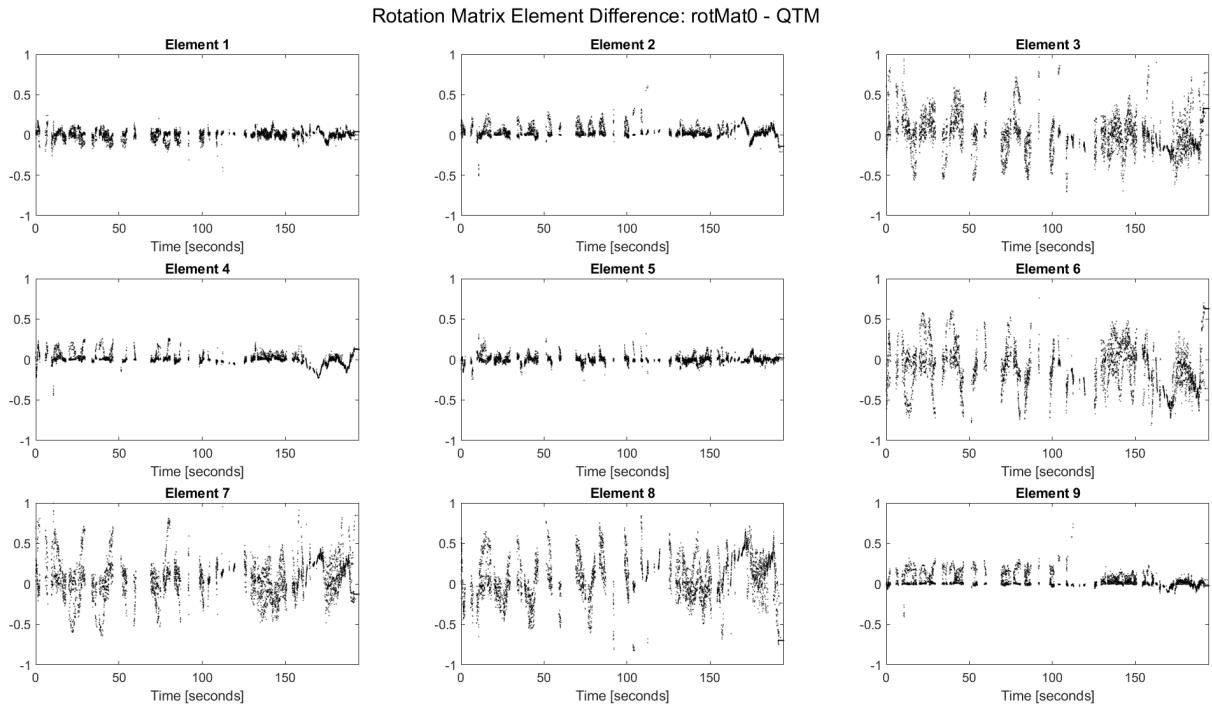


Figure E.26: Difference between QTM and first estimate's rotation matrix elements

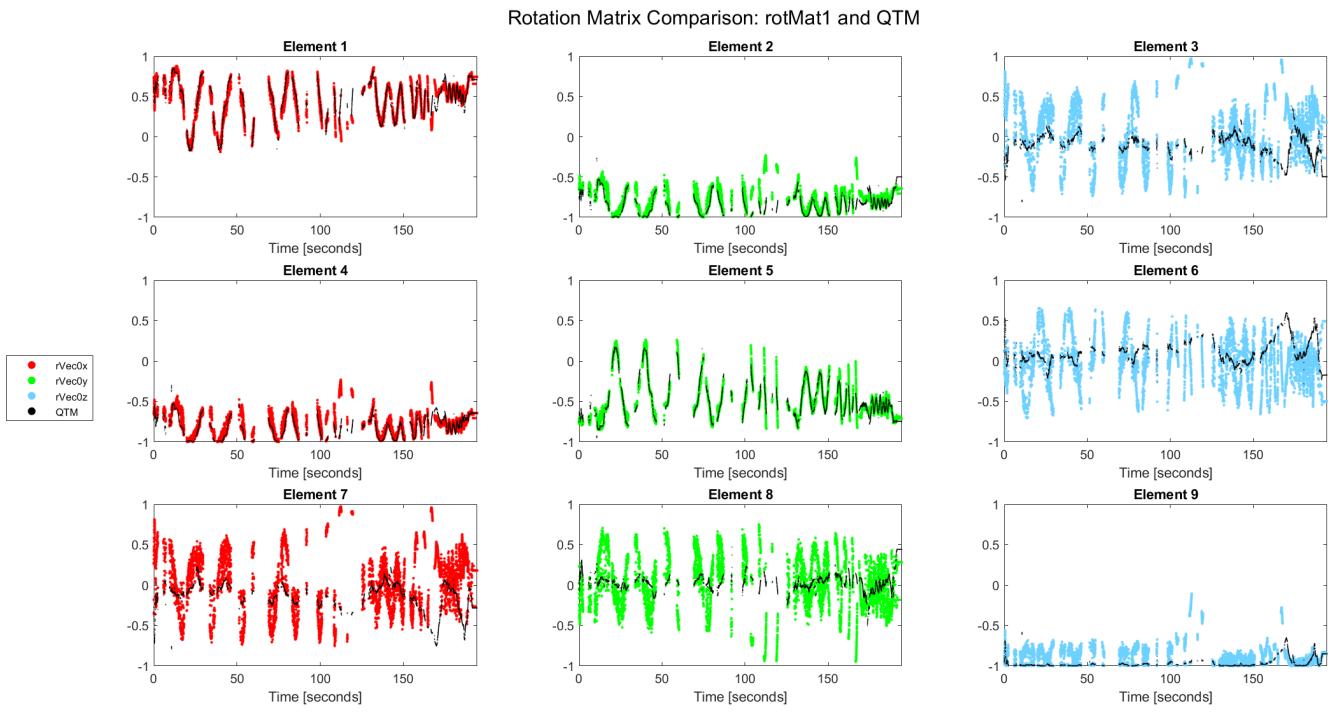


Figure E.27: Comparison of QTM and second estimate's rotation matrix elements

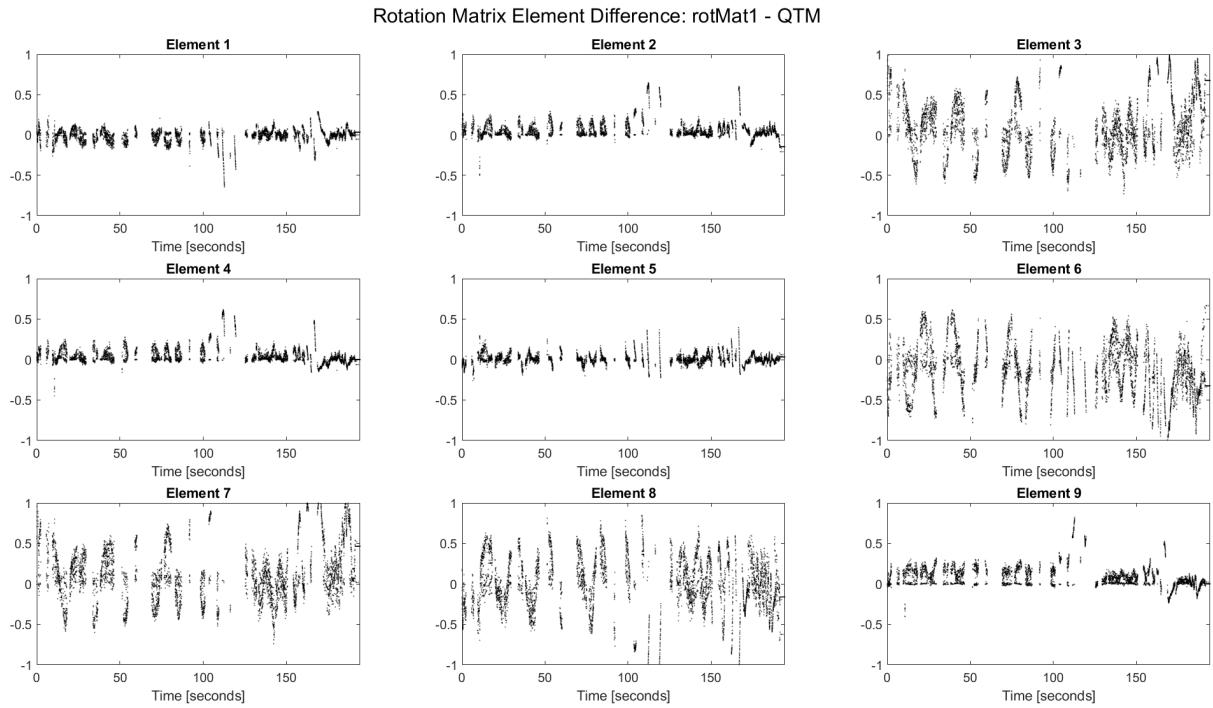


Figure E.28: Difference between QTM and second estimate's rotation matrix elements

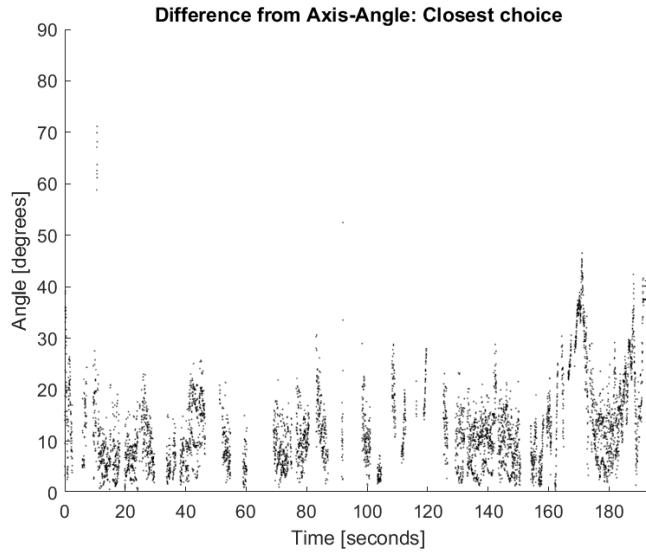


Figure E.29: Axis-Angle plot of angular difference between QTM and closest estimate

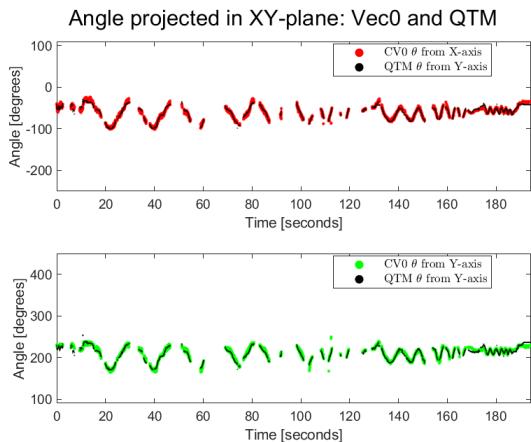


Figure E.30: XY-projected angle between global and first estimated axes, compared to QTM

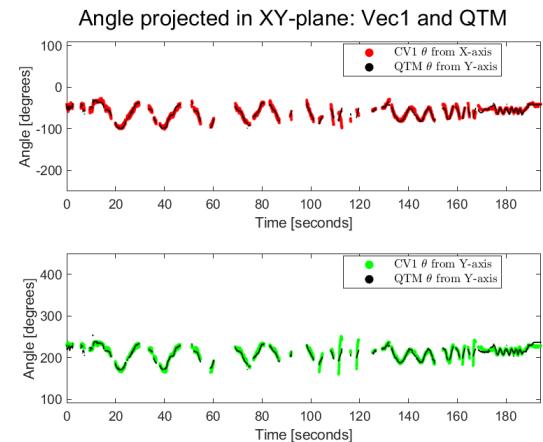


Figure E.31: XY-projected angle between global and second estimated axes, compared to QTM

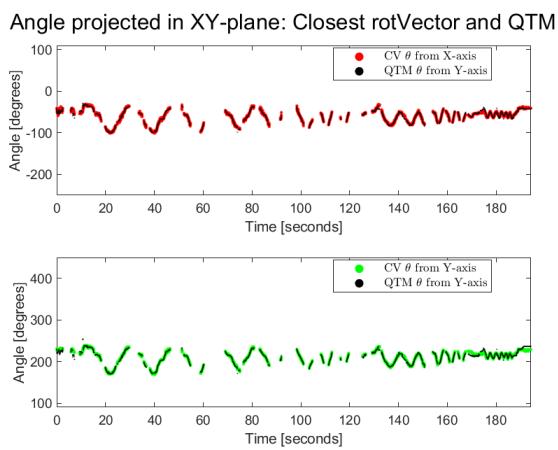


Figure E.32: XY-projected angle between global and chosen estimated axes, compared to QTM

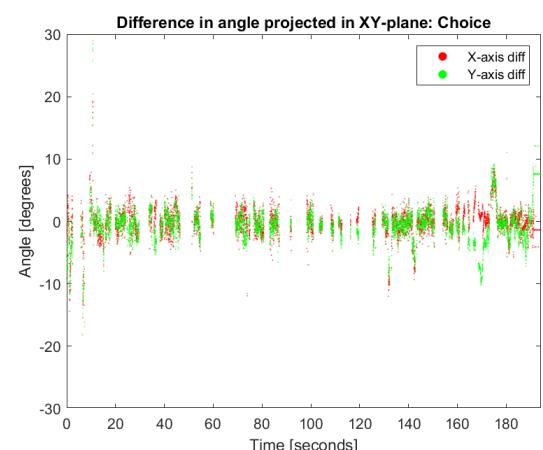


Figure E.33: XY-projected angle difference between chosen estimate and QTM axes

### E.3.4 Test #3 Results

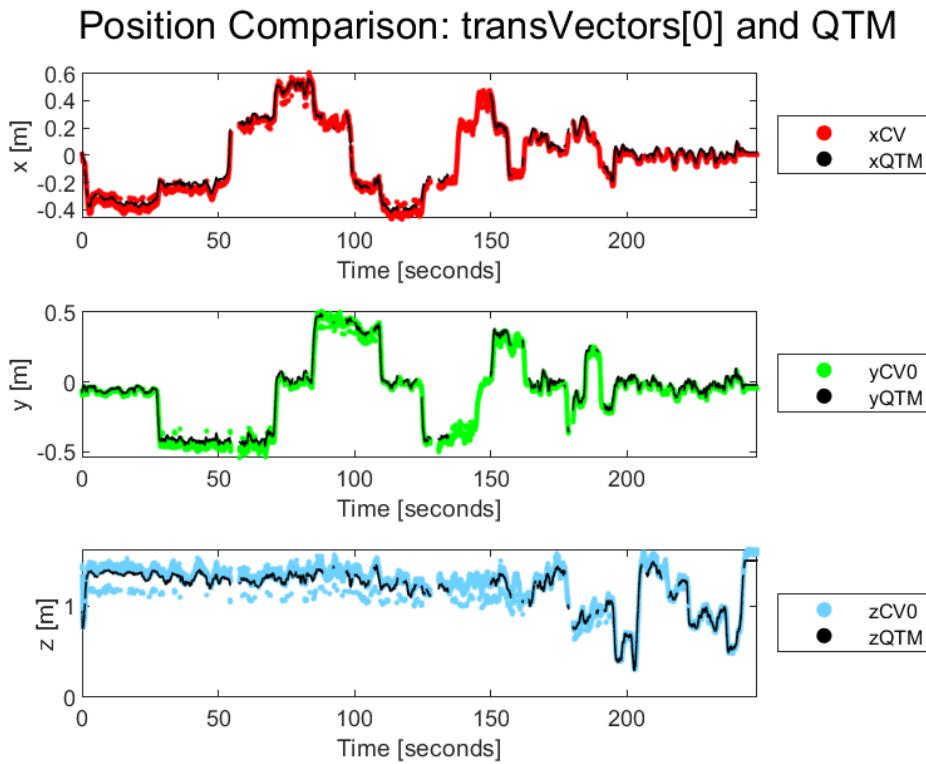


Figure E.34: Position plot, comparing elements of translation vectors

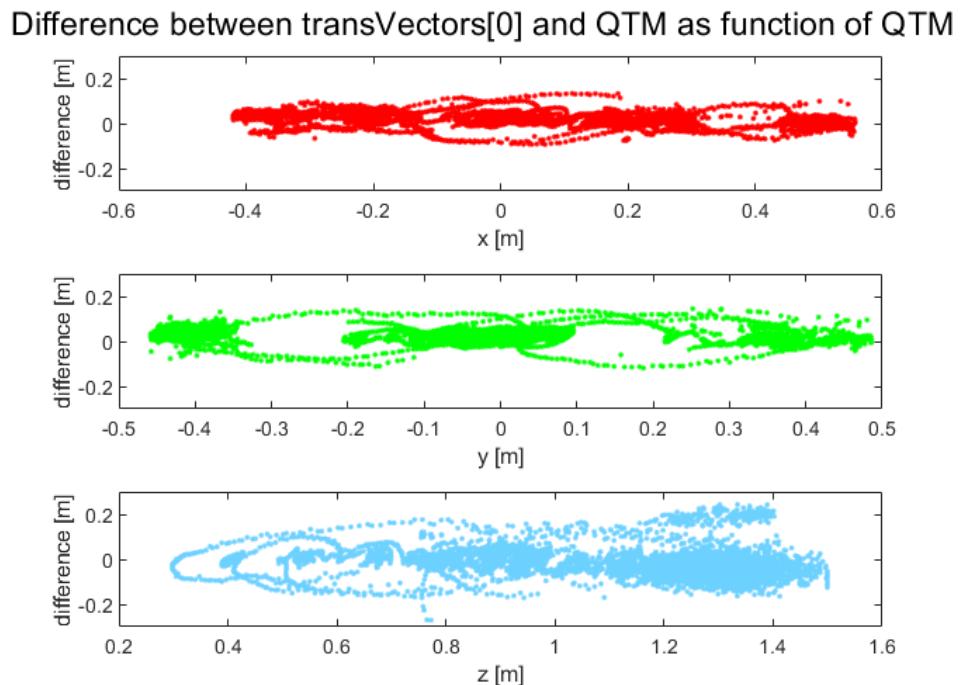


Figure E.35: Difference between first estimated translation and QTM, as function of QTM position

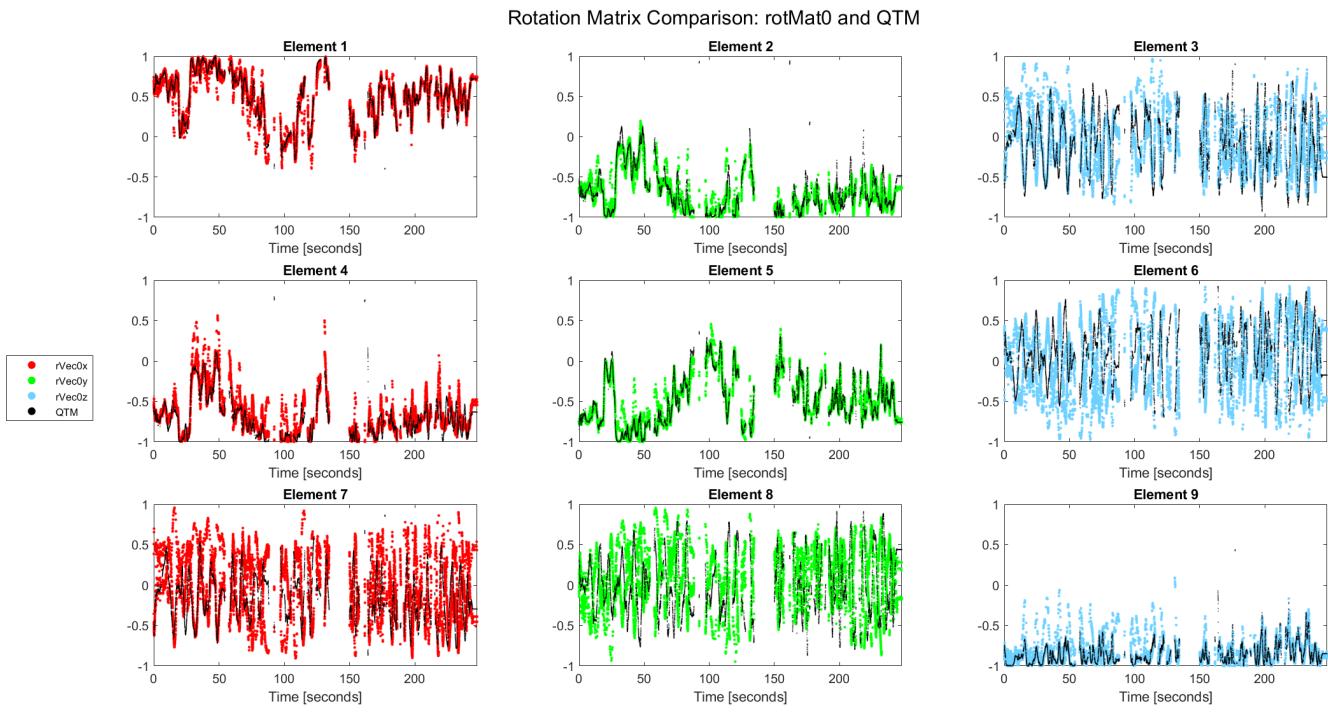


Figure E.36: Comparison of QTM and first estimate's rotation matrix elements

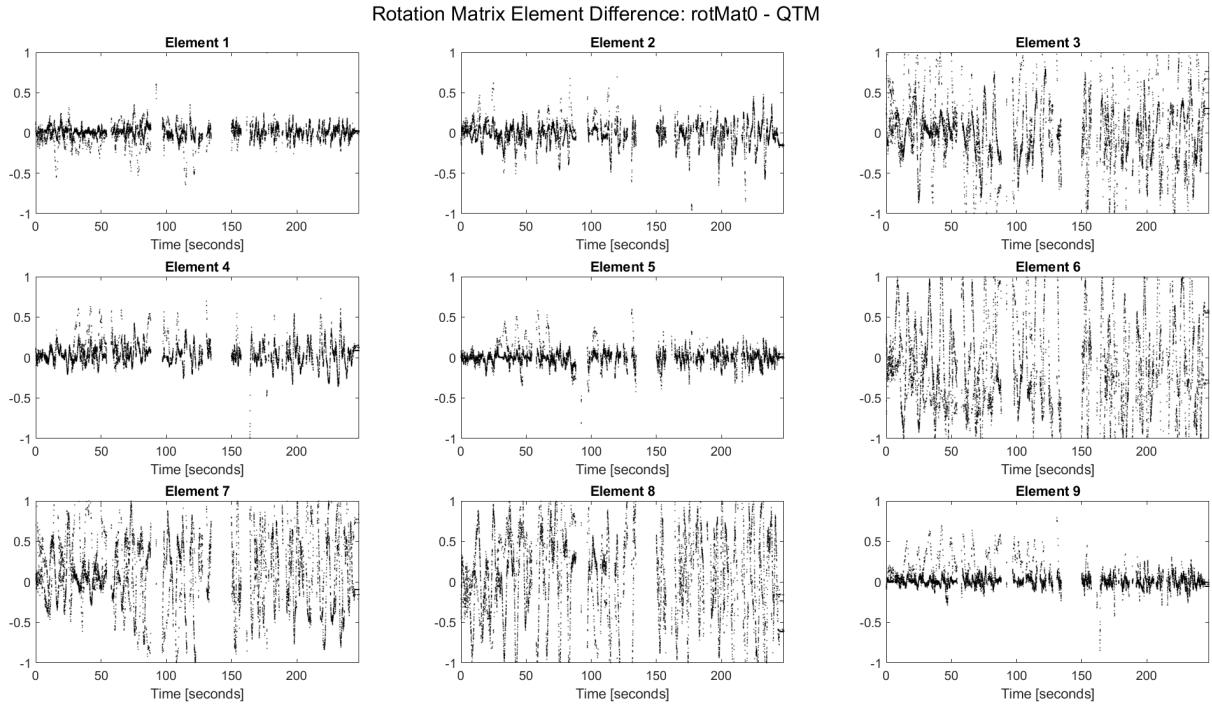


Figure E.37: Difference between QTM and first estimate's rotation matrix elements

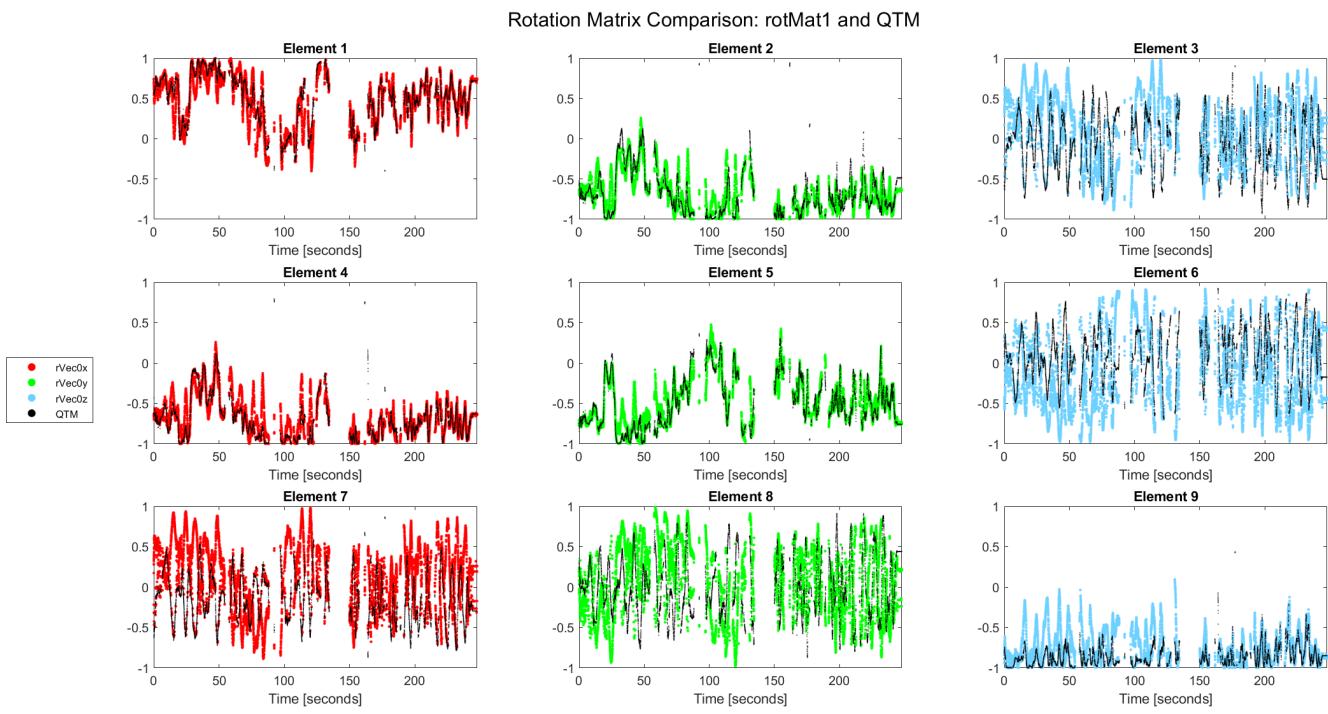


Figure E.38: Comparison of QTM and second estimate's rotation matrix elements

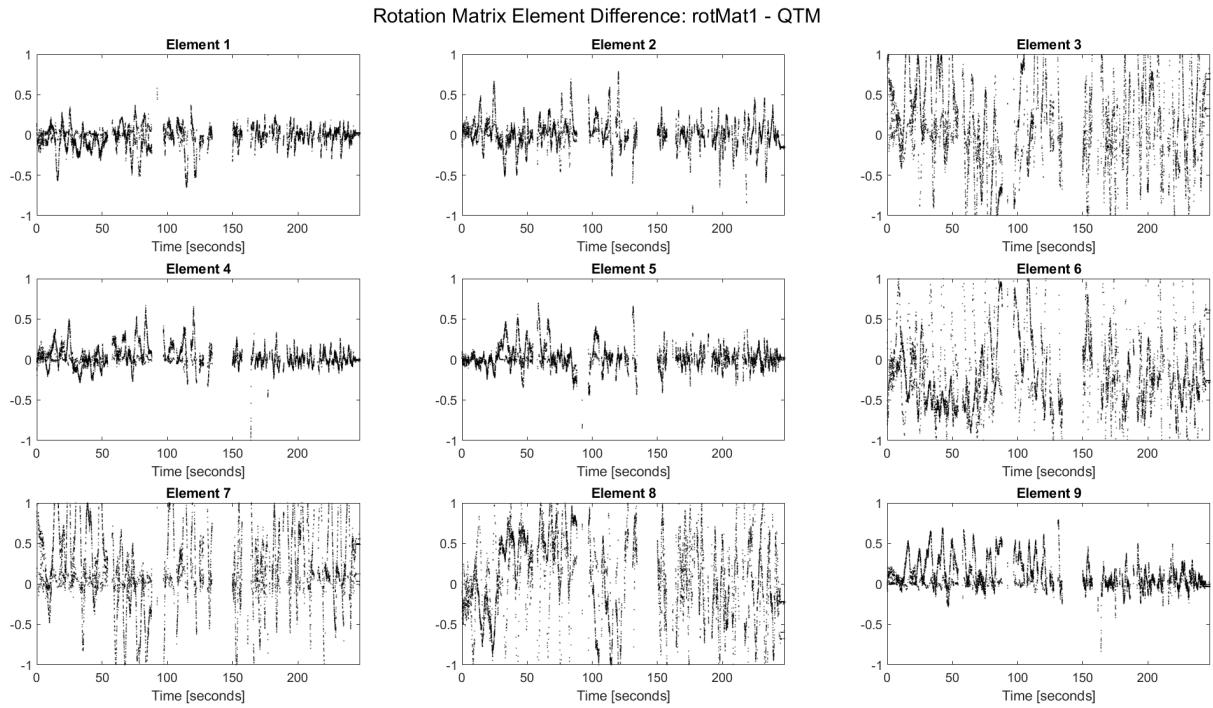


Figure E.39: Difference between QTM and second estimate's rotation matrix elements

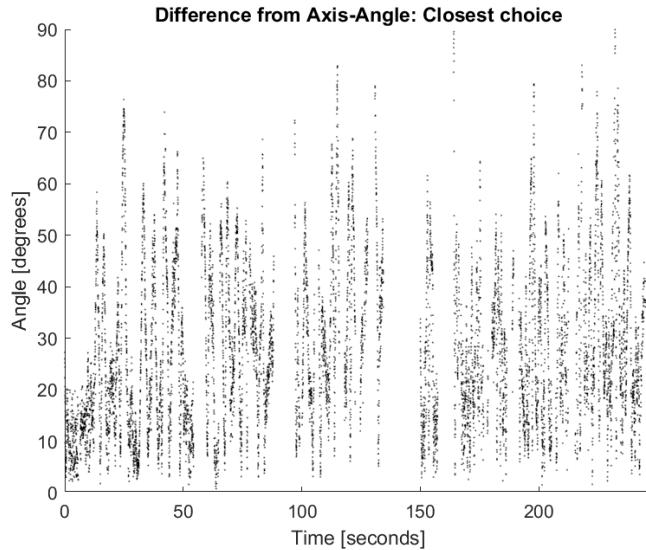


Figure E.40: Axis-Angle plot of angular difference between QTM and closest estimate

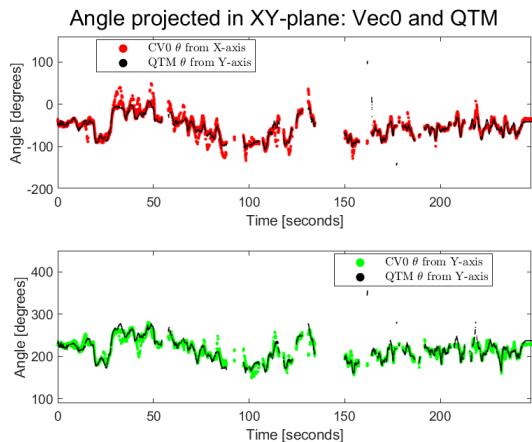


Figure E.41: XY-projected angle between global and first estimated axes, compared to QTM

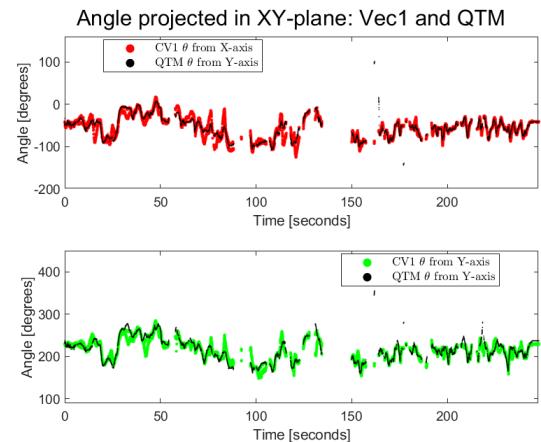


Figure E.42: XY-projected angle between global and second estimated axes, compared to QTM

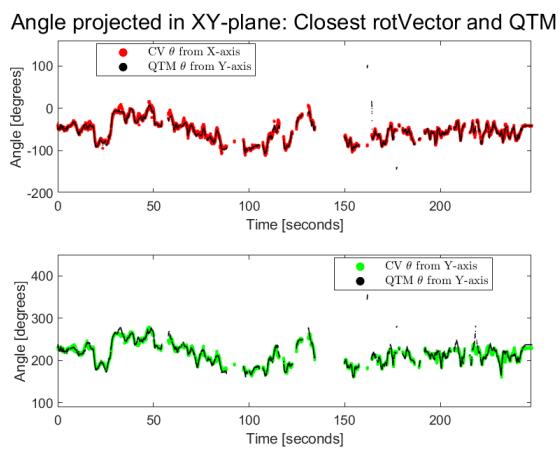


Figure E.43: XY-projected angle between global and chosen estimated axes, compared to QTM

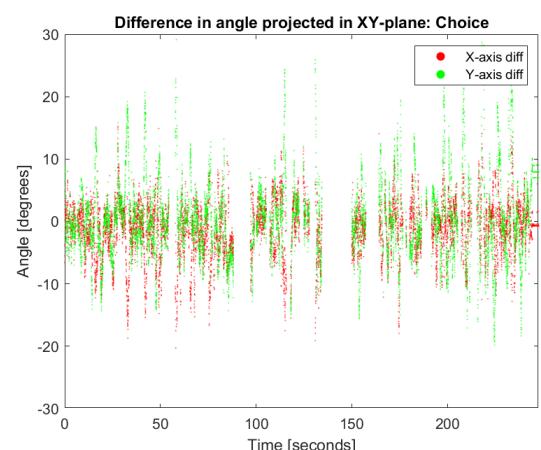


Figure E.44: XY-projected angle difference between chosen estimate and QTM axes

### E.3.5 Test #4 Results

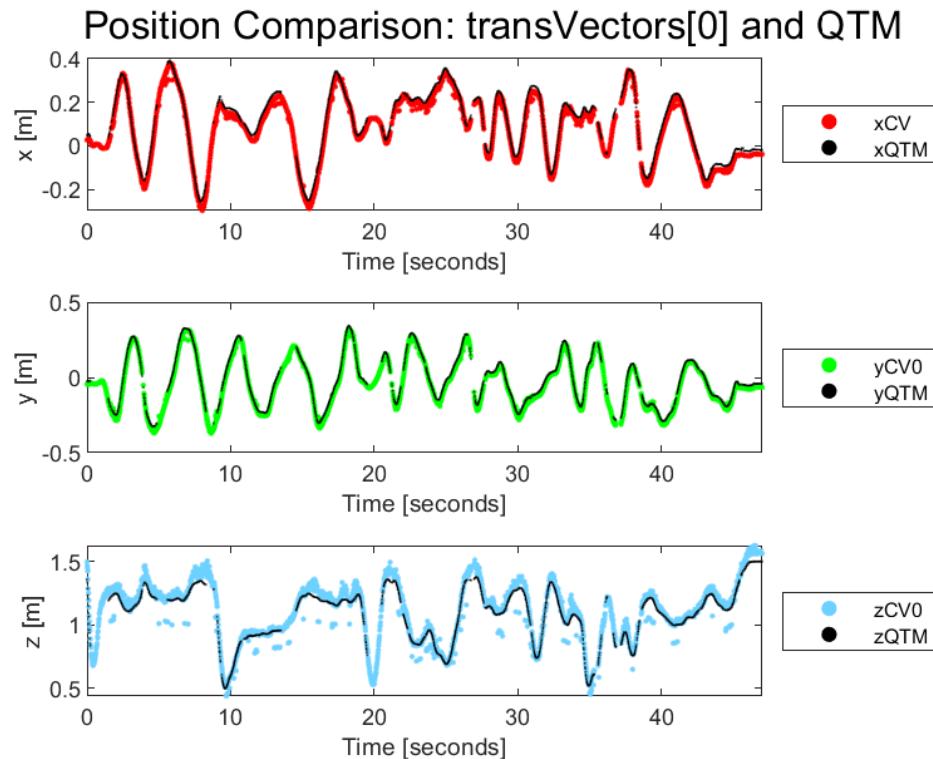


Figure E.45: Position plot, comparing elements of translation vectors

### Difference between transVectors[0] and QTM as function of QTM

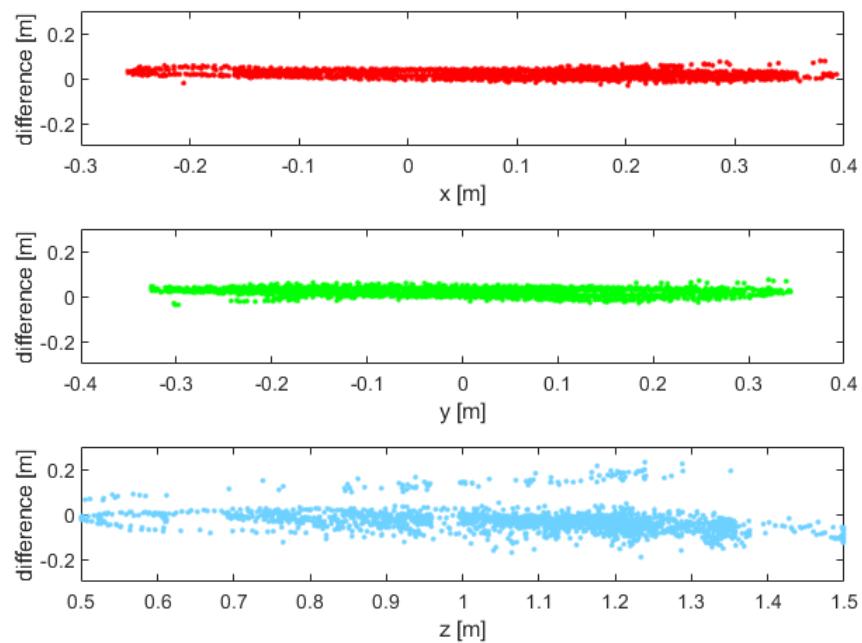


Figure E.46: Difference between first estimated translation and QTM, as function of QTM position

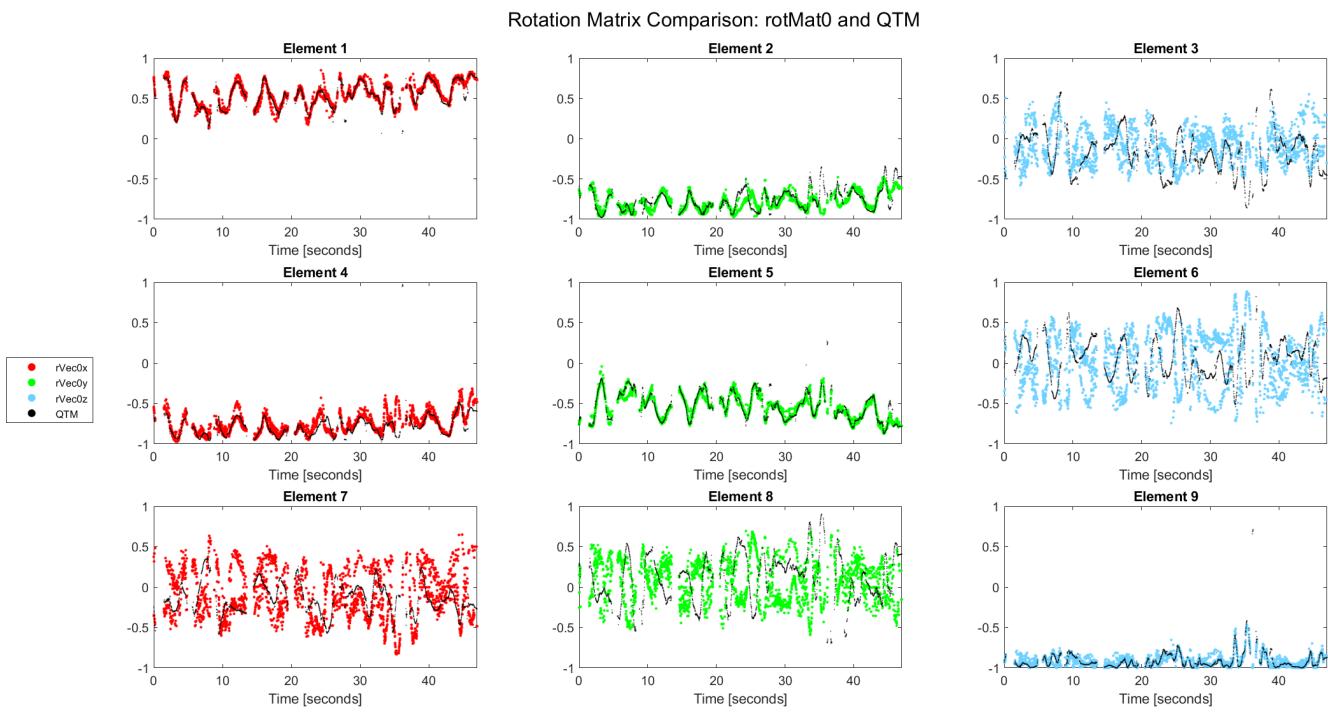


Figure E.47: Comparison of QTM and first estimate's rotation matrix elements

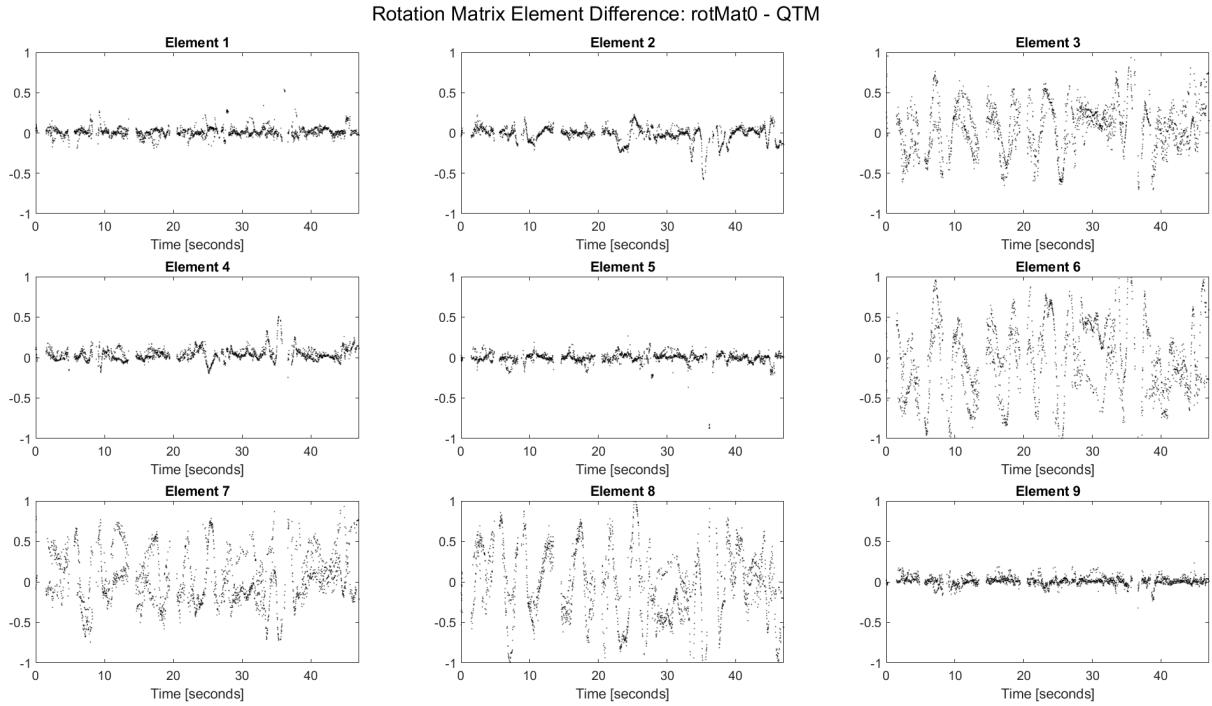


Figure E.48: Difference between QTM and first estimate's rotation matrix elements

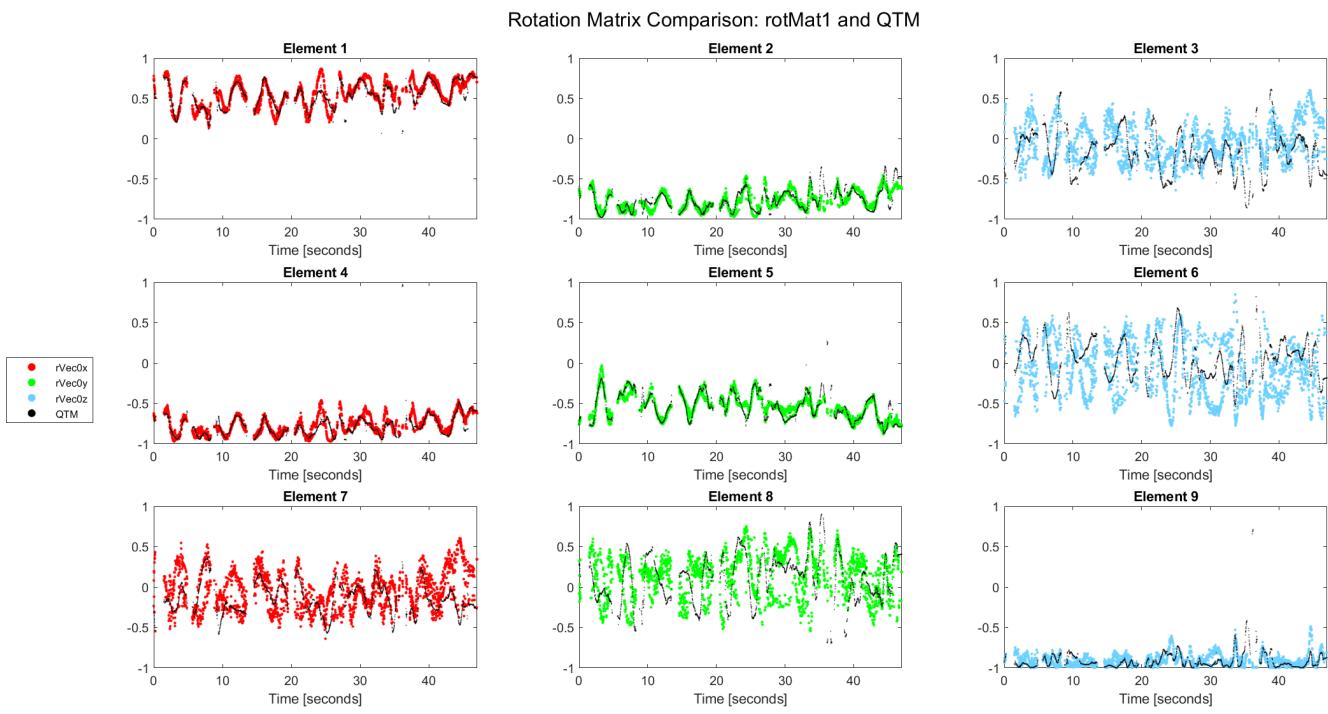


Figure E.49: Comparison of QTM and second estimate's rotation matrix elements

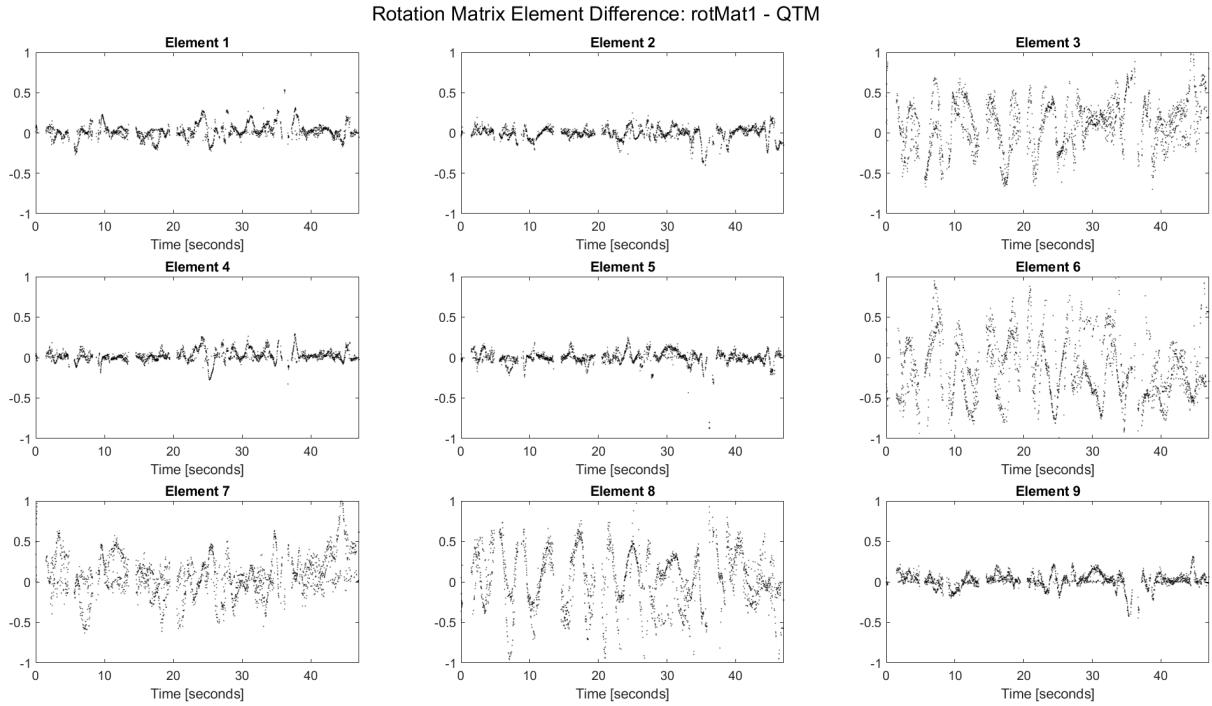


Figure E.50: Difference between QTM and second estimate's rotation matrix elements

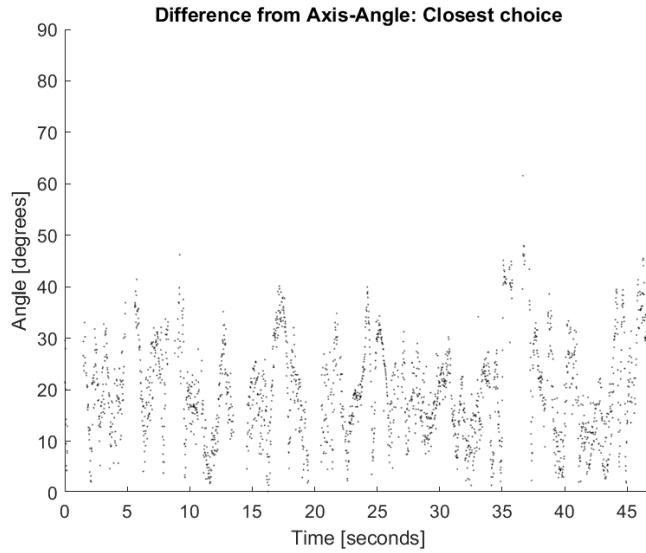


Figure E.51: Axis-Angle plot of angular difference between QTM and closest estimate

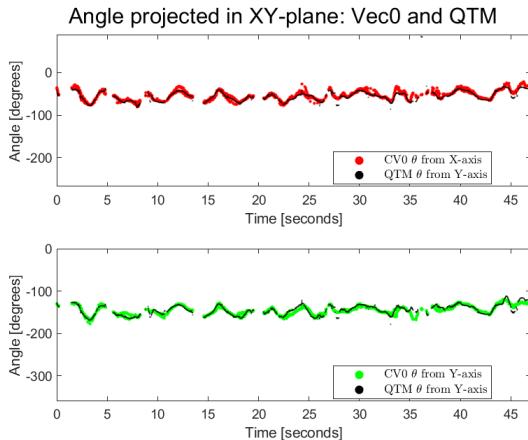


Figure E.52: XY-projected angle between global and first estimated axes, compared to QTM

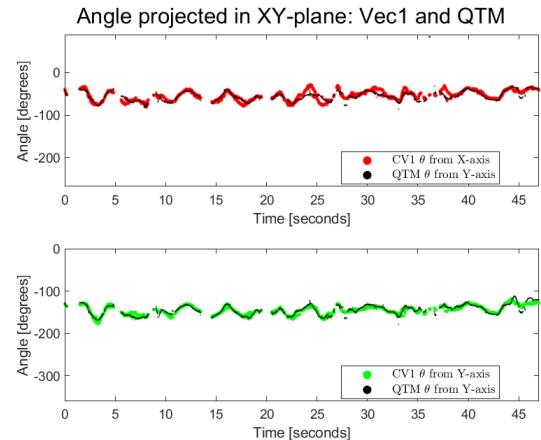


Figure E.53: XY-projected angle between global and second estimated axes, compared to QTM

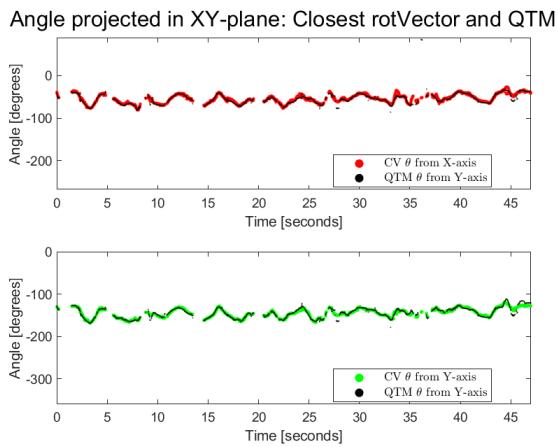


Figure E.54: XY-projected angle between global and chosen estimated axes, compared to QTM

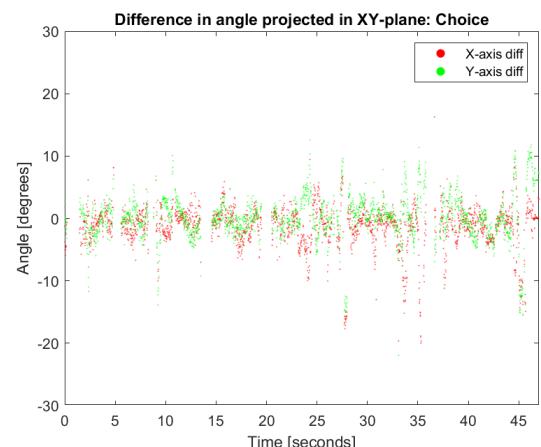


Figure E.55: XY-projected angle difference between chosen estimate and QTM axes

## E.4 1D Position Kalman Filter Results

### E.4.1 Test #0

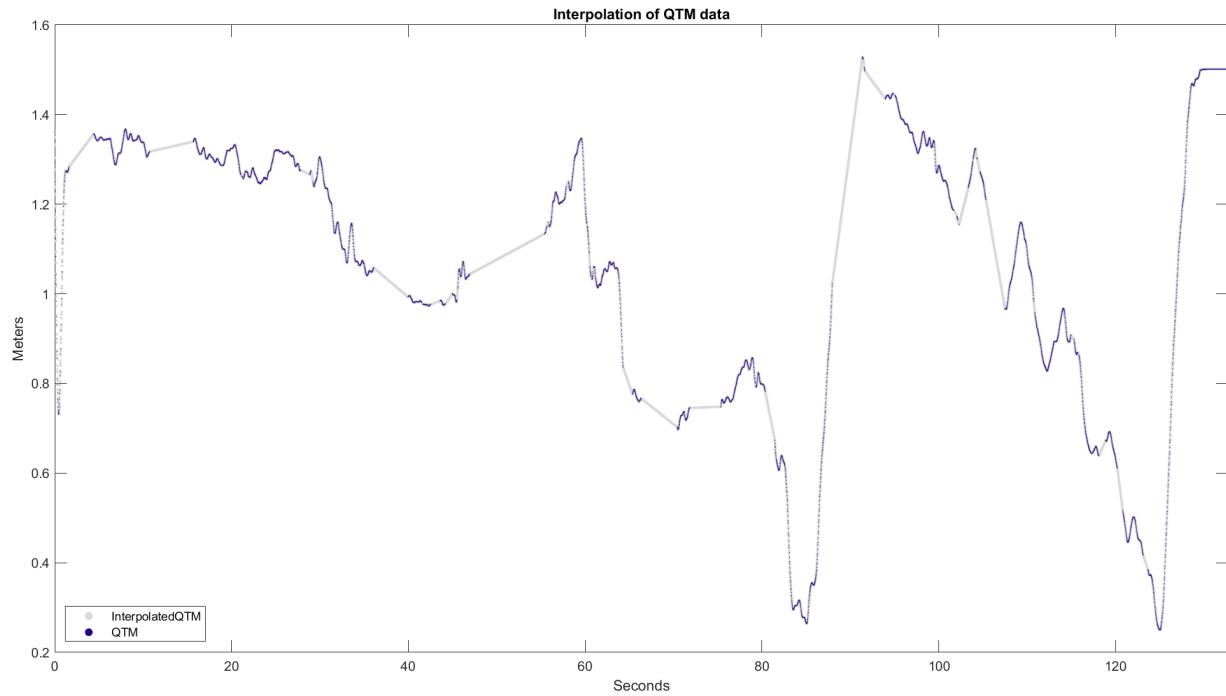


Figure E.56: Linear interpolation of QTM data

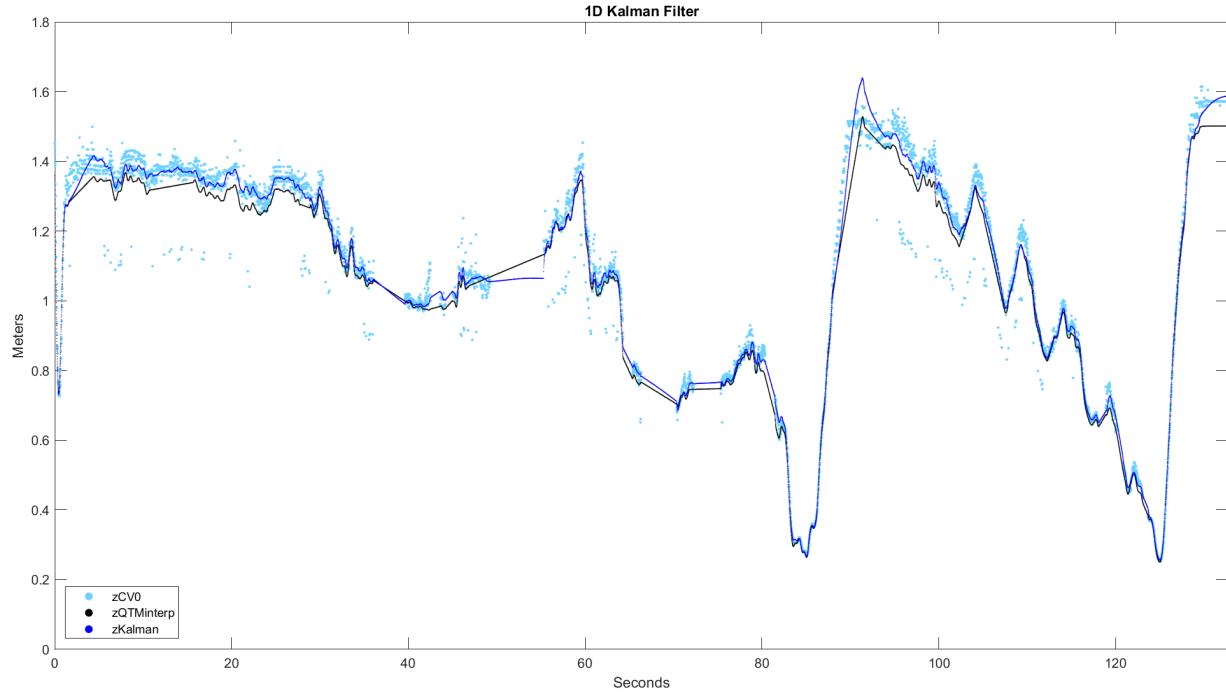


Figure E.57: 1D Kalman filter compared to measurements and QTM reference

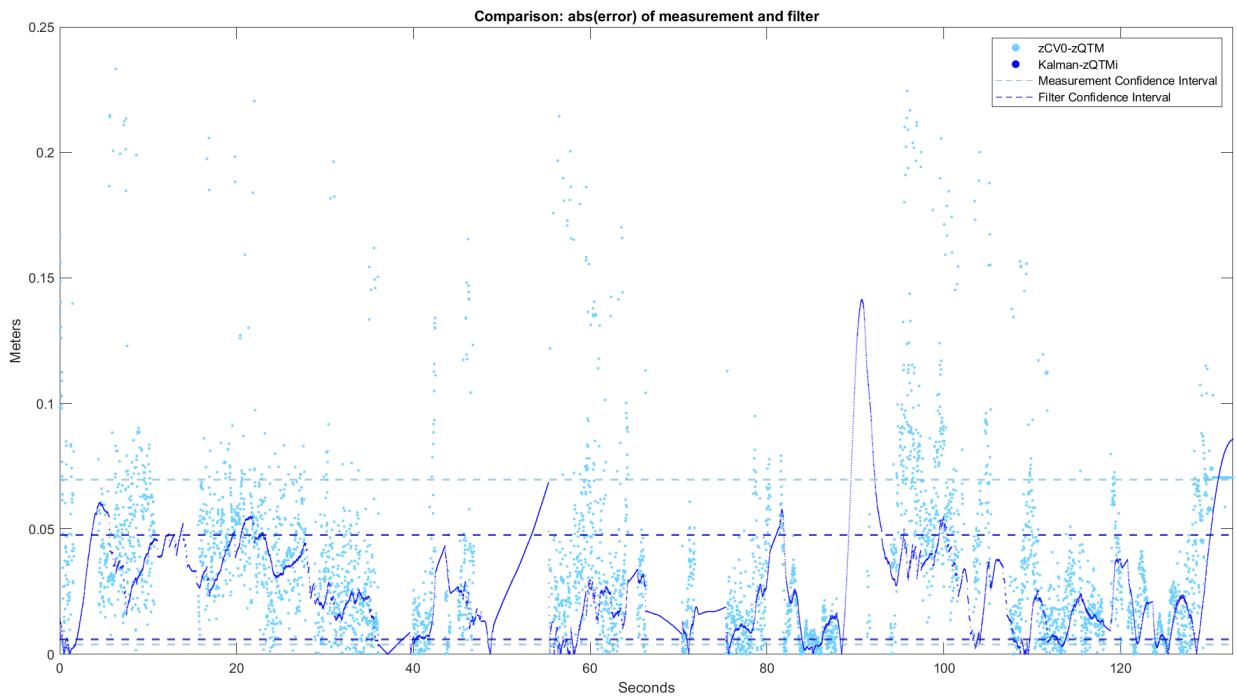


Figure E.58: Confidence intervals for absolute value of error without and with filter

#### E.4.2 Test #1

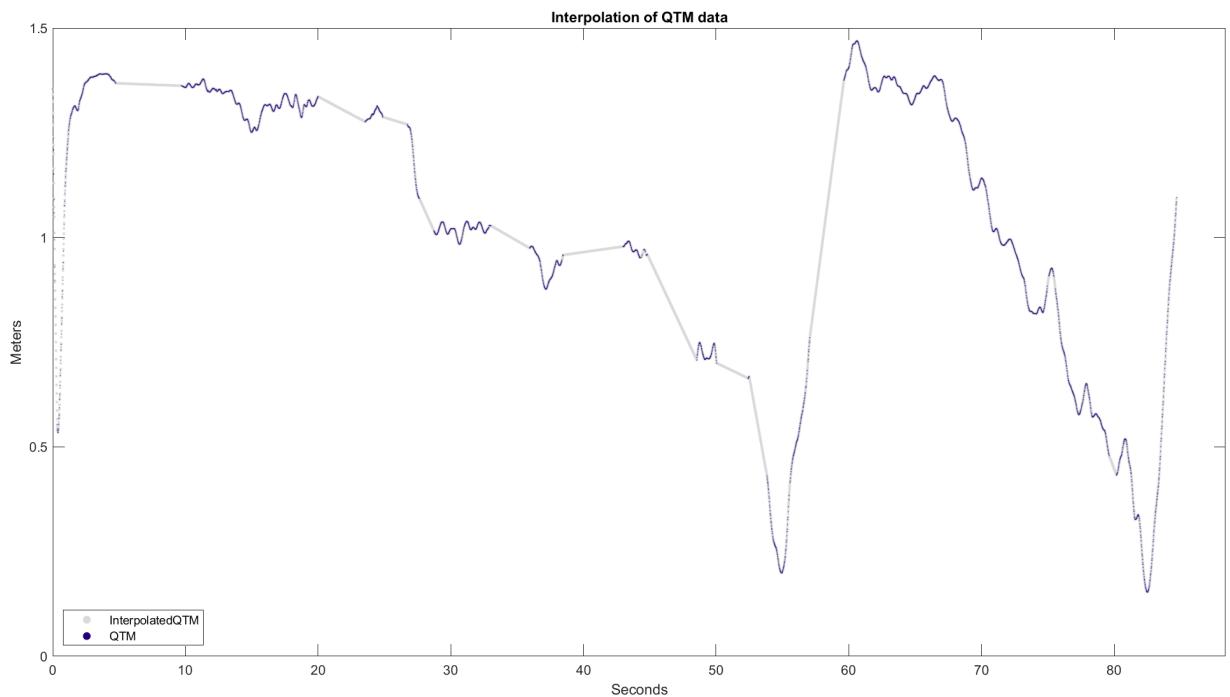


Figure E.59: Linear interpolation of QTM data

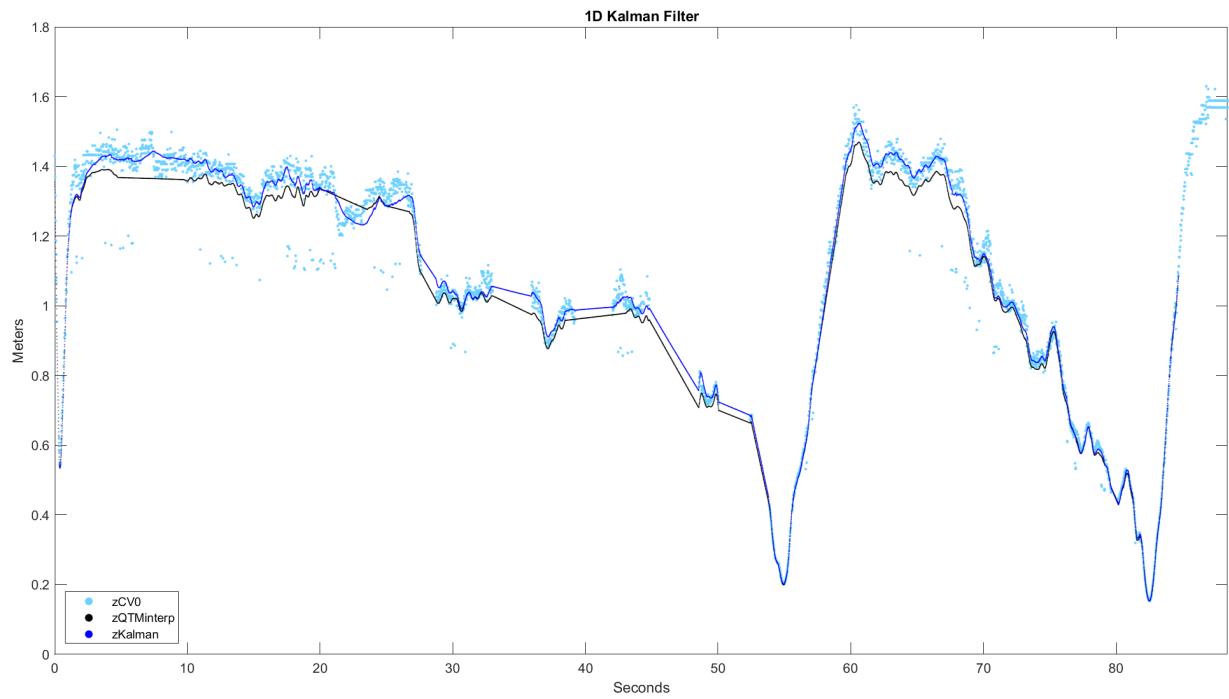


Figure E.60: 1D Kalman filter compared to measurements and QTM reference

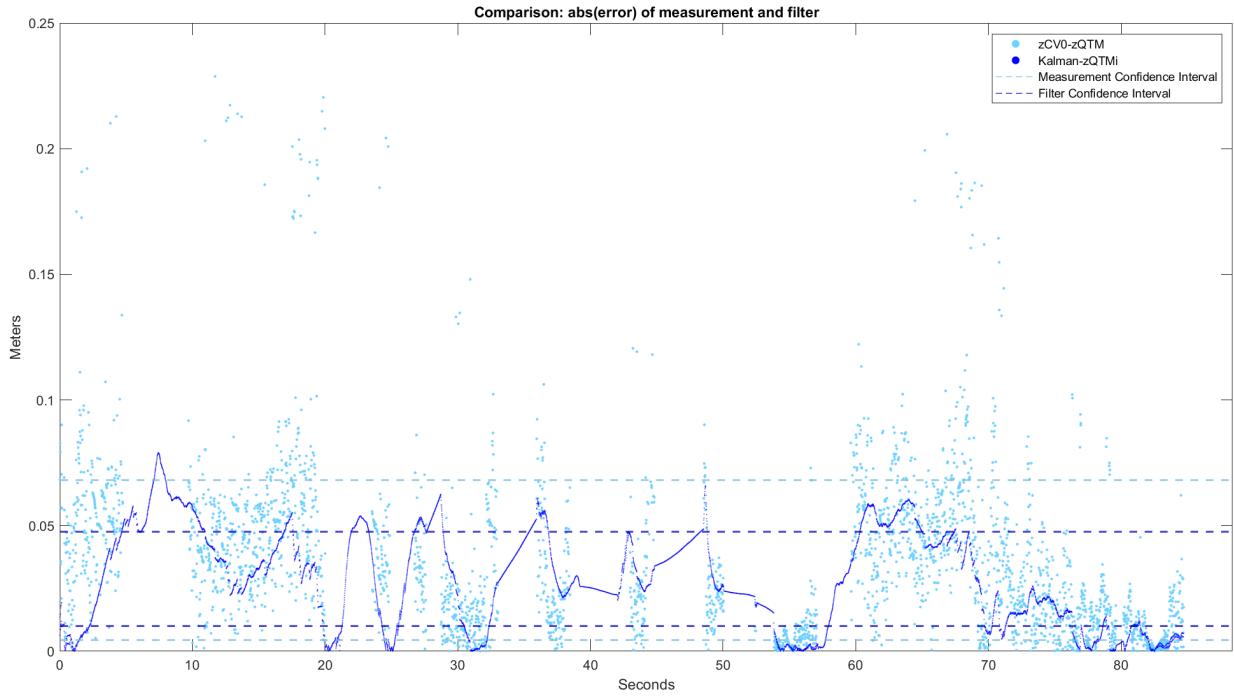


Figure E.61: Confidence intervals for absolute value of error without and with filter

### E.4.3 Test #2

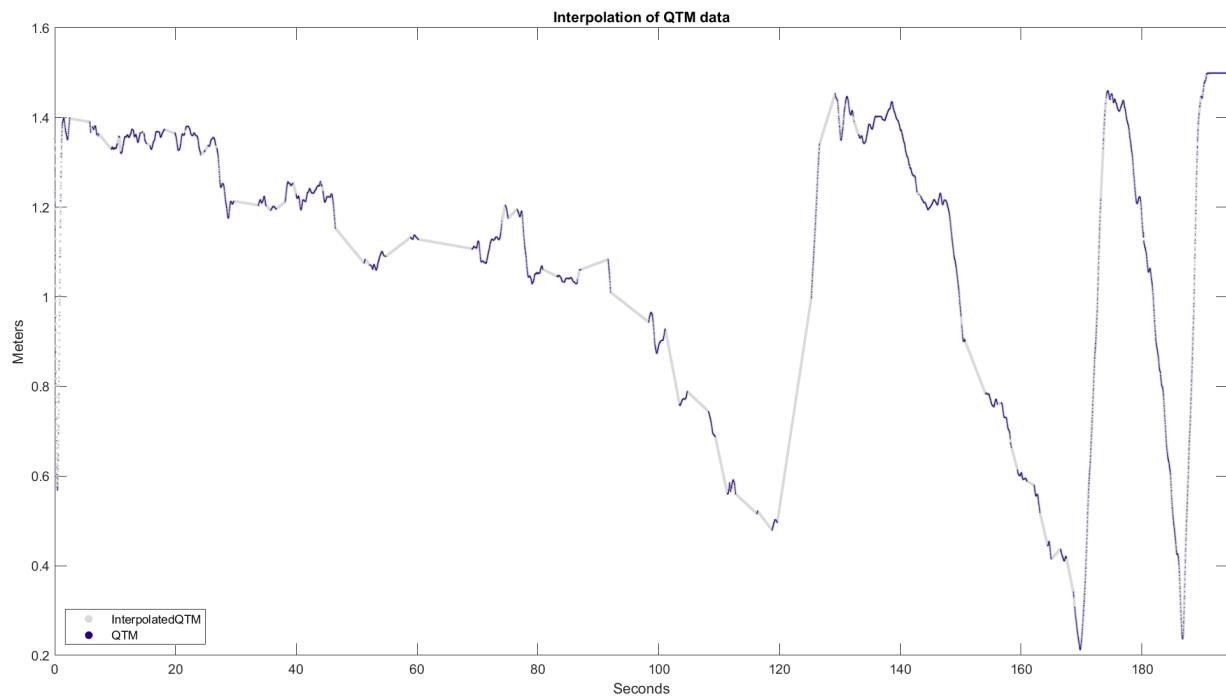


Figure E.62: Linear interpolation of QTM data

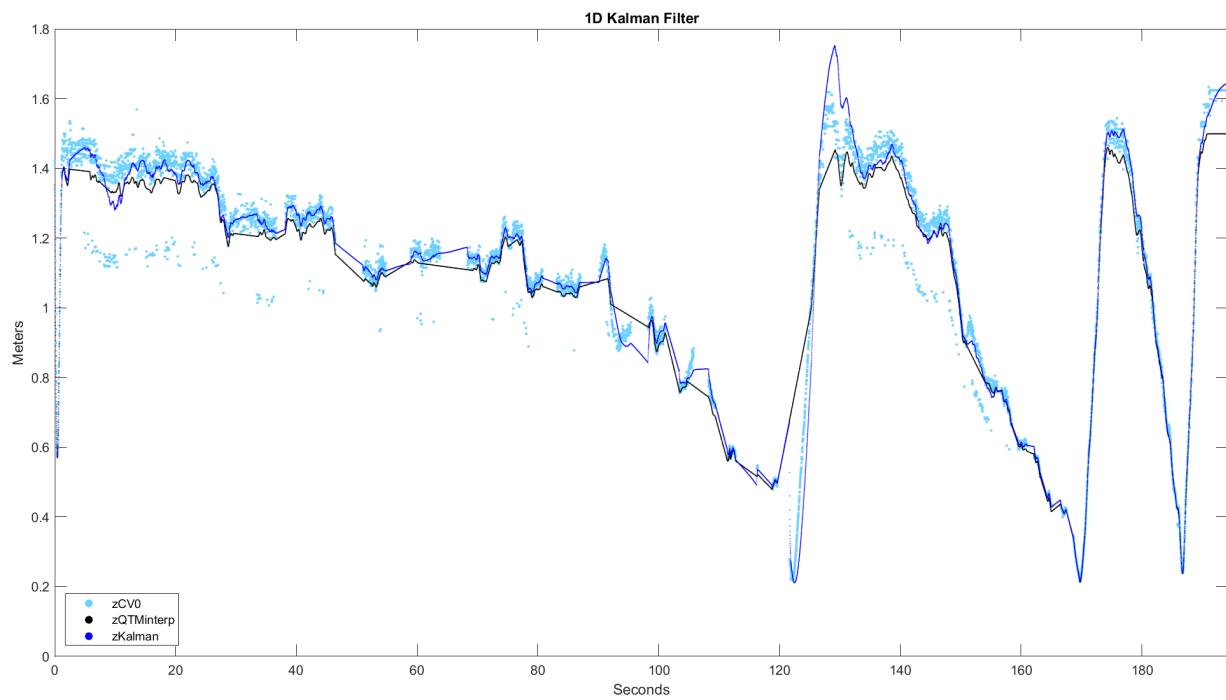


Figure E.63: 1D Kalman filter compared to measurements and QTM reference

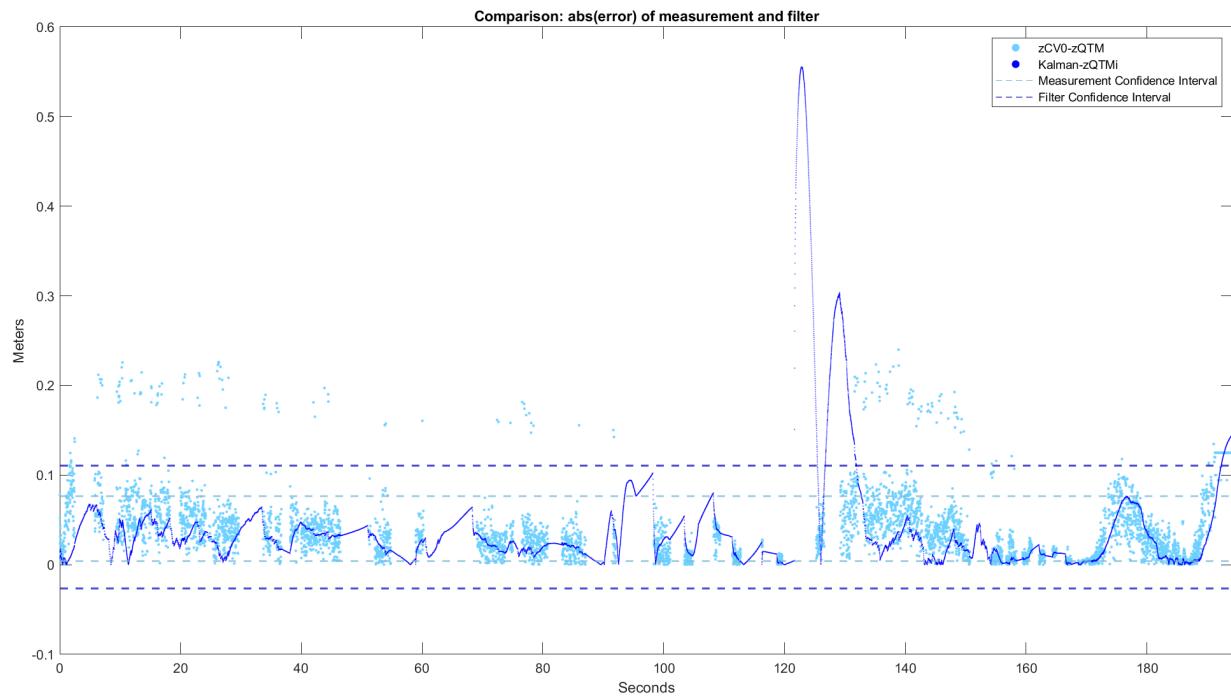


Figure E.64: Confidence intervals for absolute value of error without and with filter

#### E.4.4 Test #3

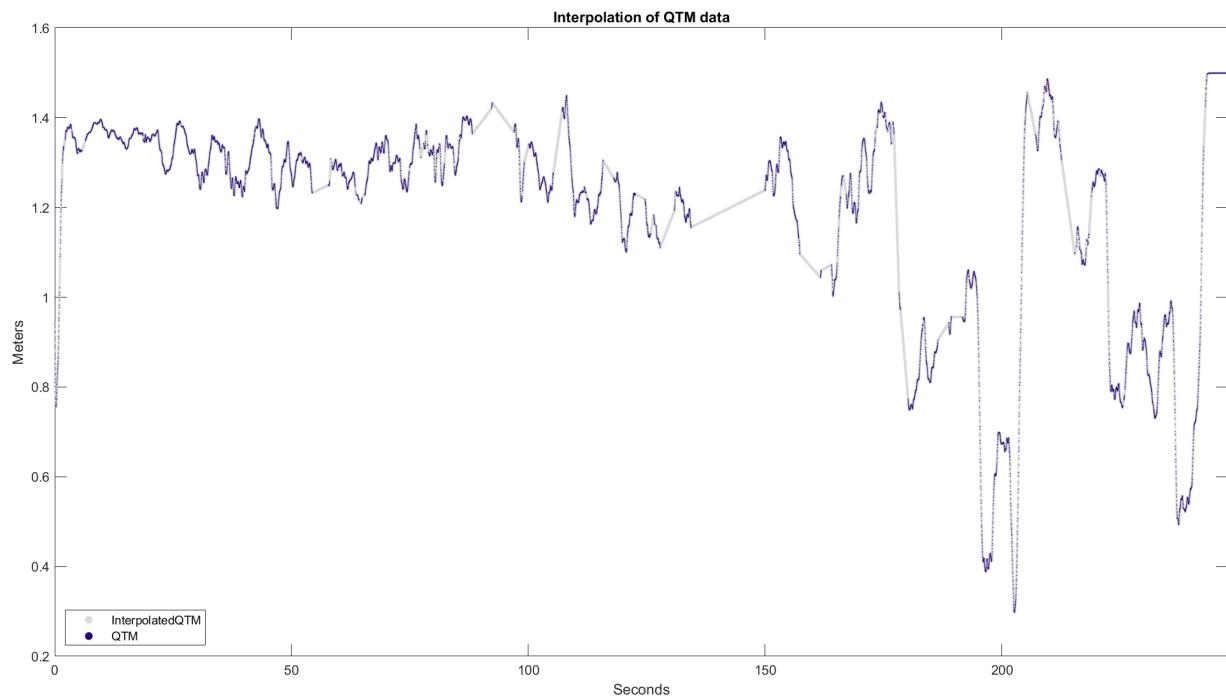


Figure E.65: Linear interpolation of QTM data

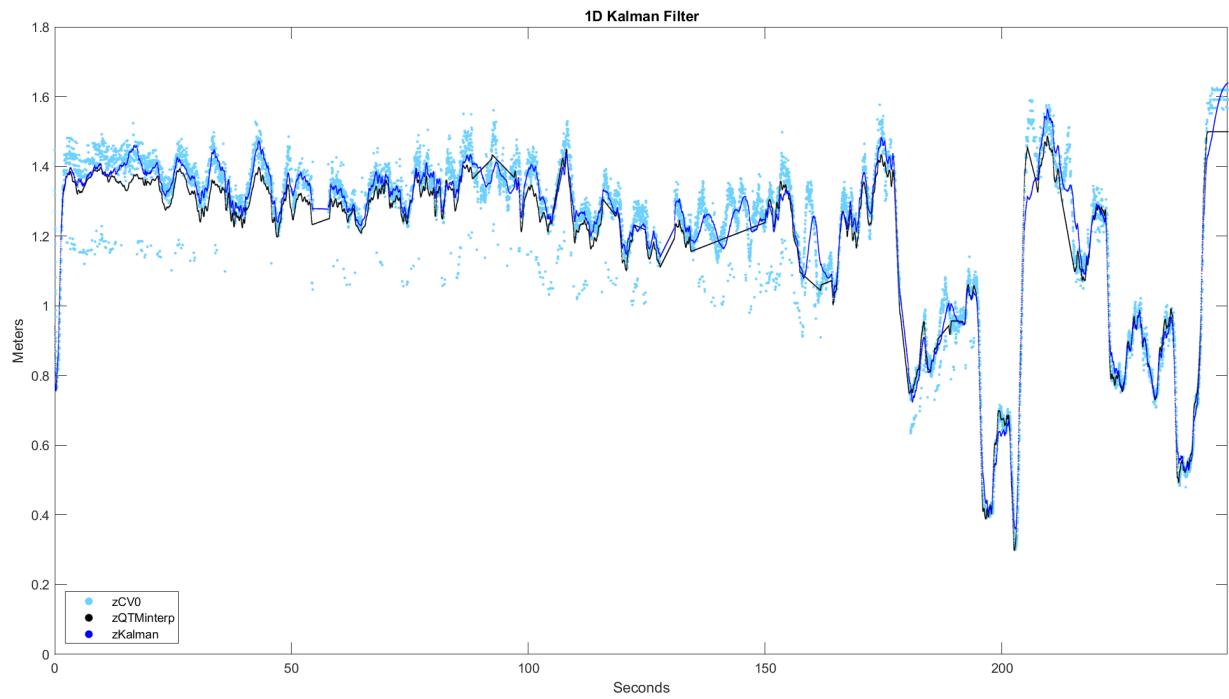


Figure E.66: 1D Kalman filter compared to measurements and QTM reference

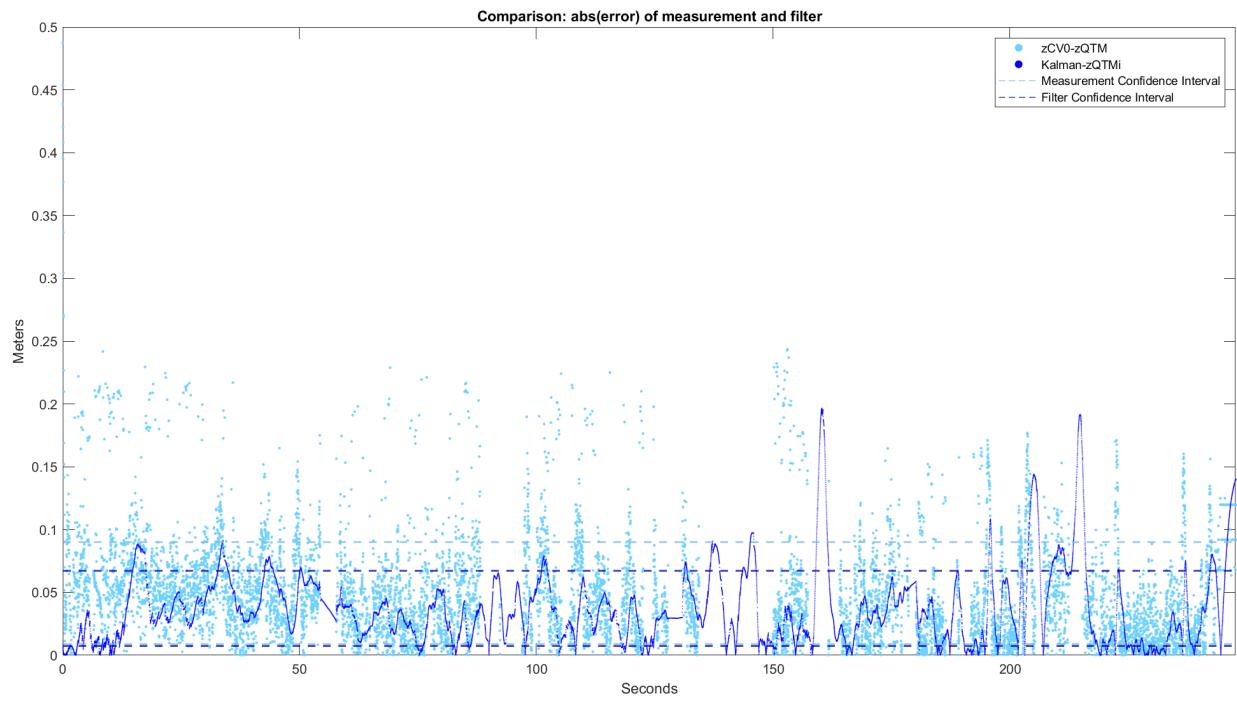


Figure E.67: Confidence intervals for absolute value of error without and with filter

#### E.4.5 Test #4

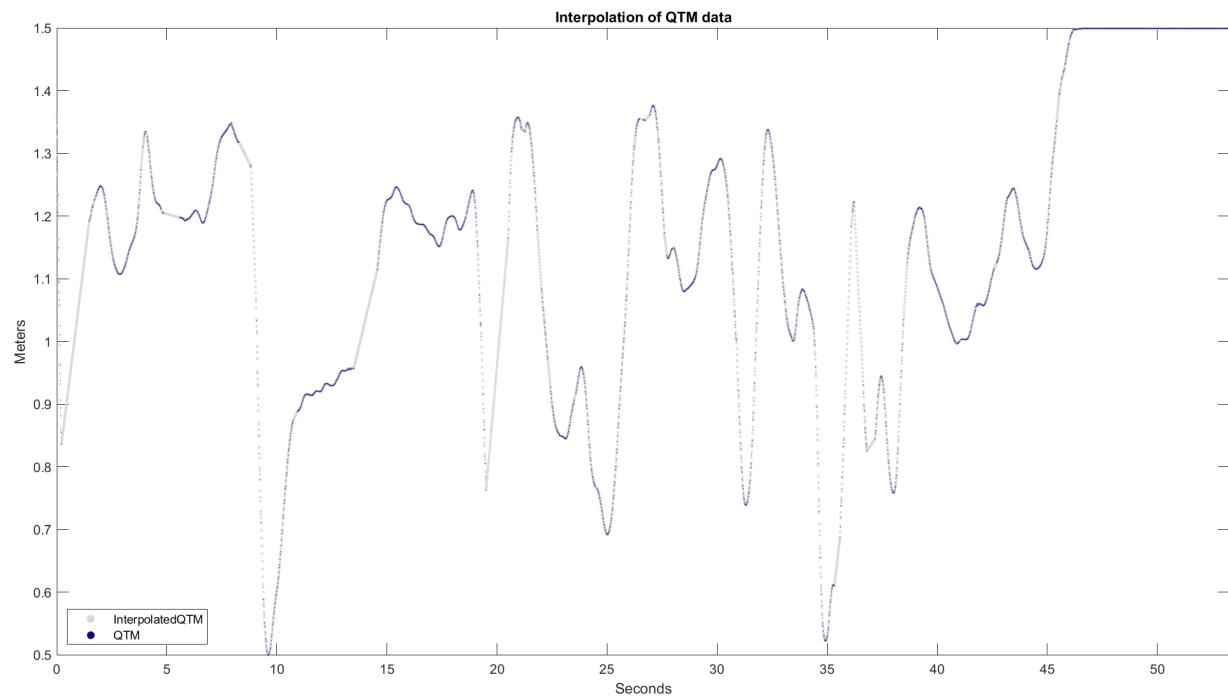


Figure E.68: Linear interpolation of QTM data

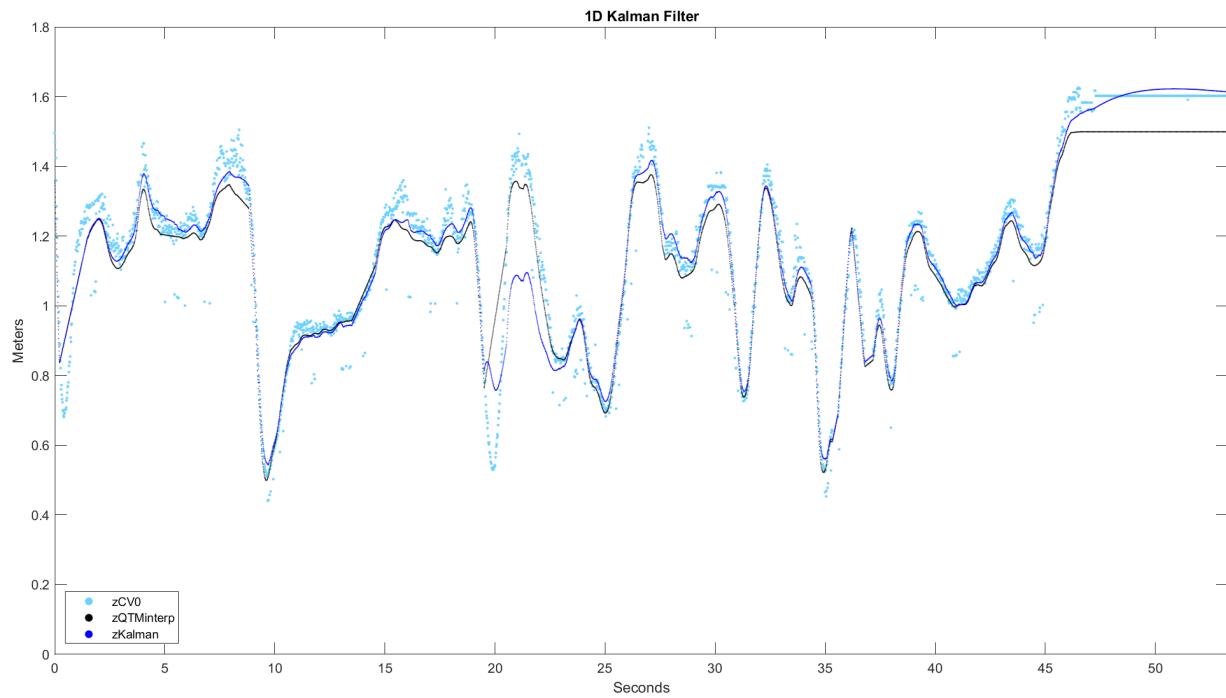


Figure E.69: 1D Kalman filter compared to measurements and QTM reference

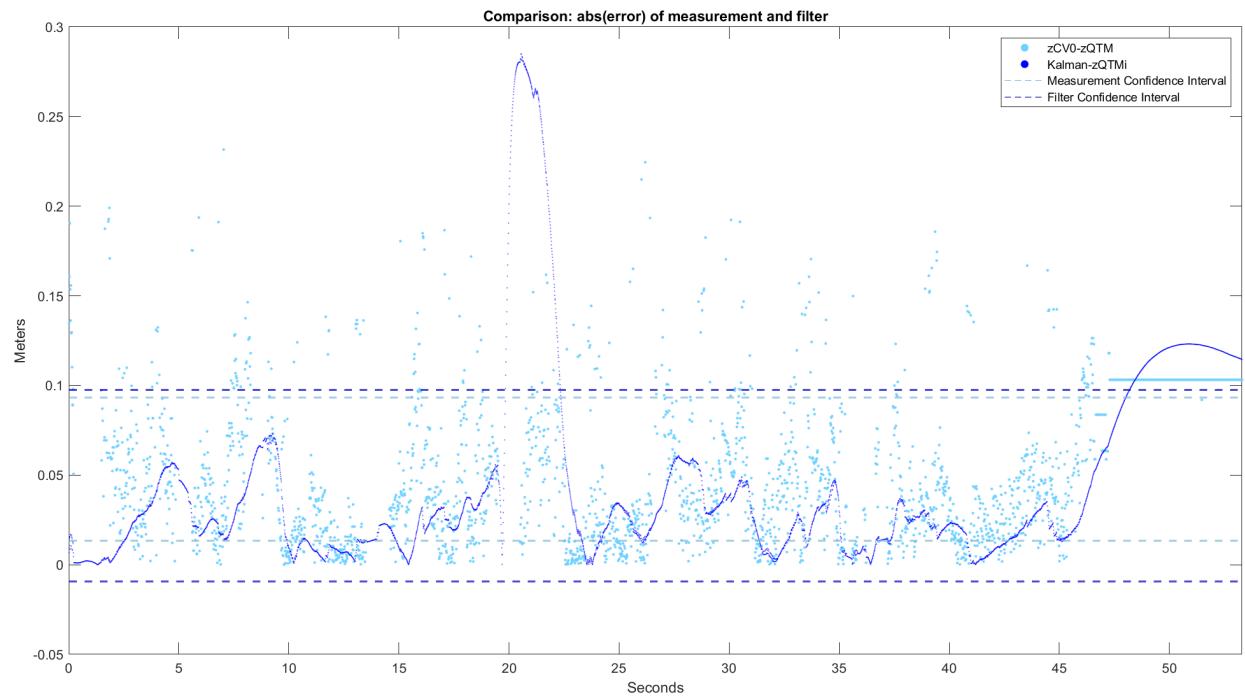


Figure E.70: Confidence intervals for absolute value of error without and with filter

## E.5 3D Position Kalman Filter Results

### E.5.1 Test #0

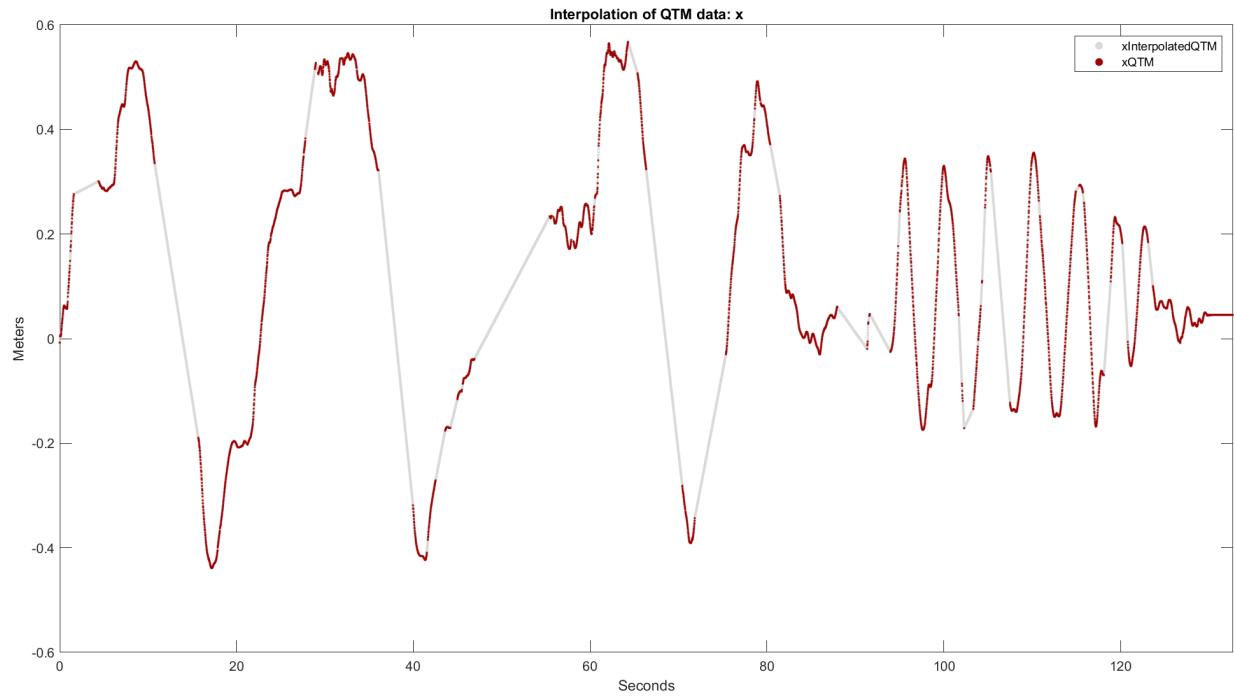


Figure E.71: X interpolation of QTM data

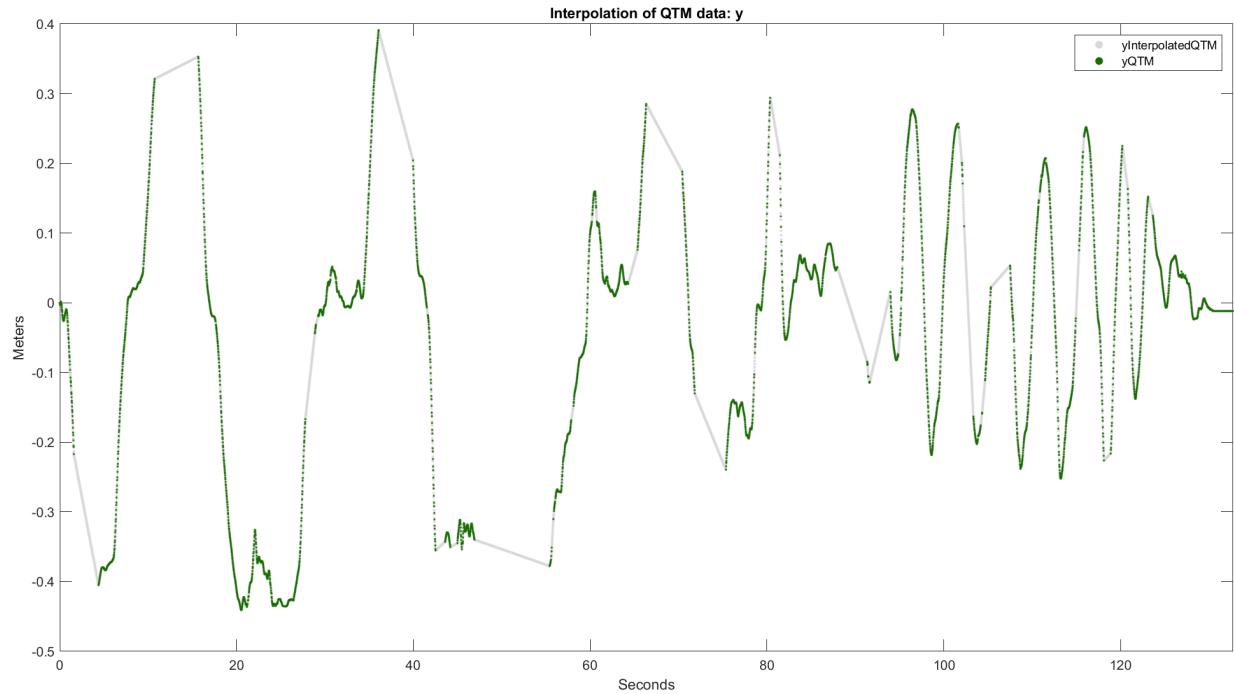


Figure E.72: Y interpolation of QTM data

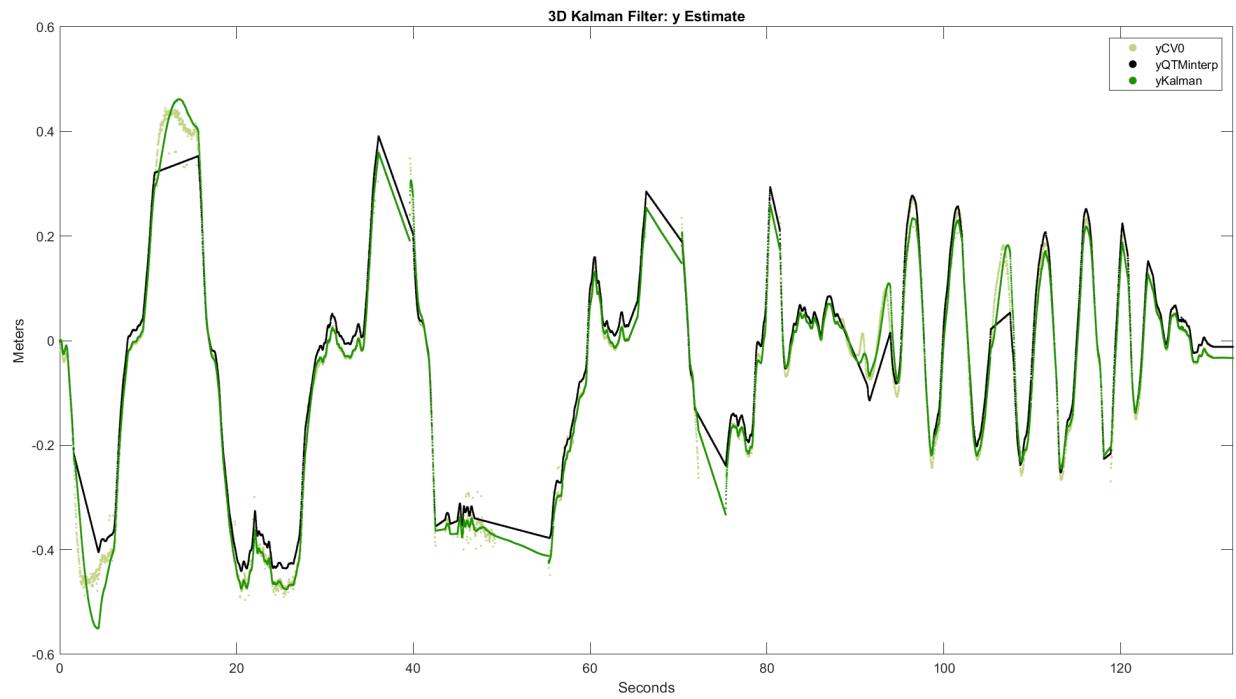


Figure E.73: Comparison of x estimate, measurements and reference

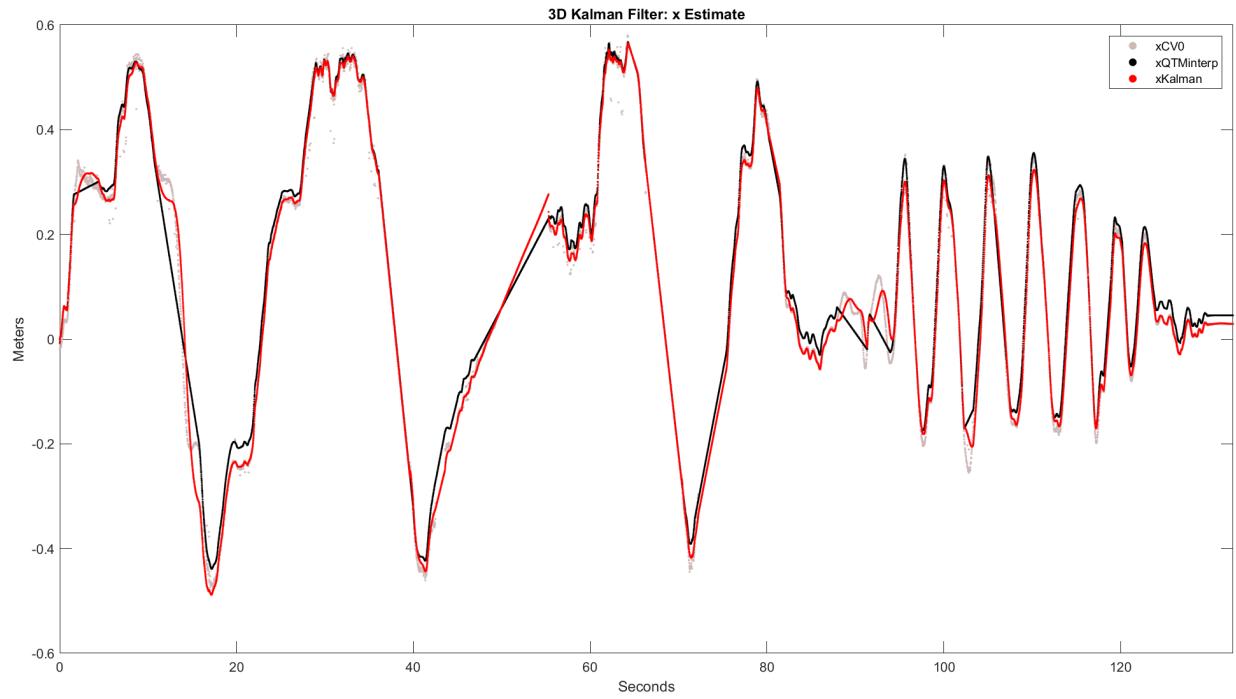


Figure E.74: Comparison of y estimate, measurements and reference

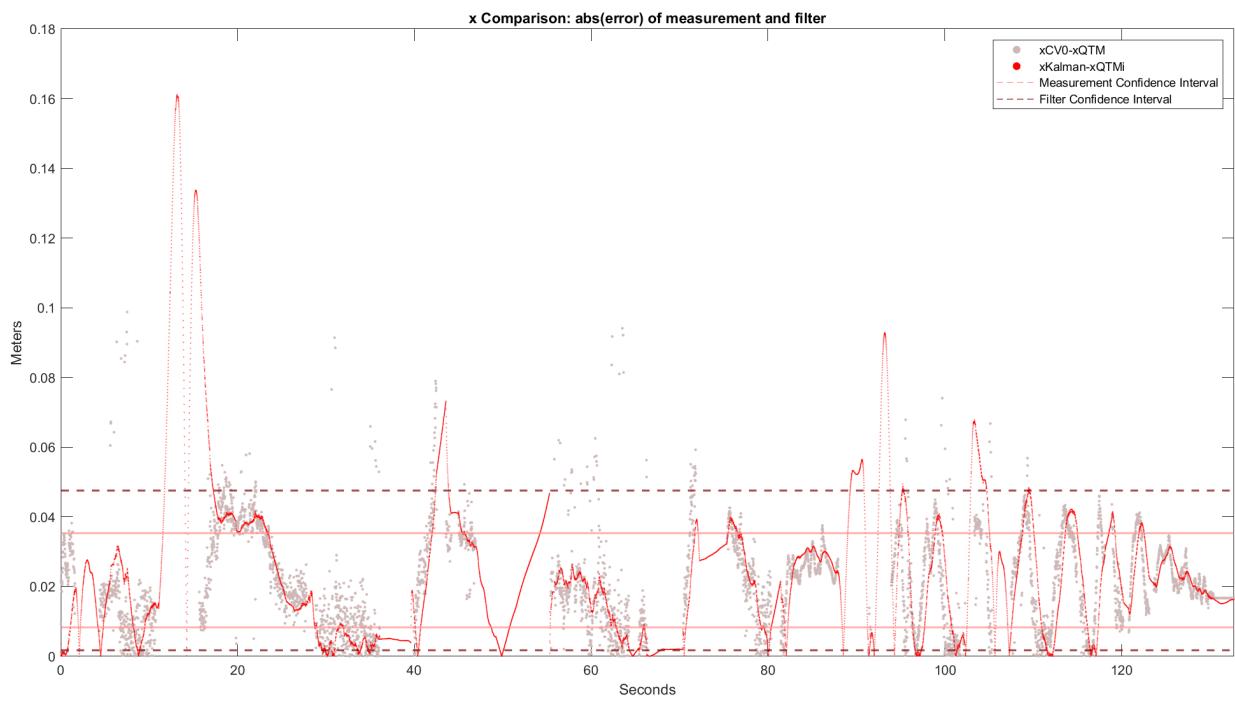


Figure E.75: Confidence intervals for absolute value of x error without and with filter

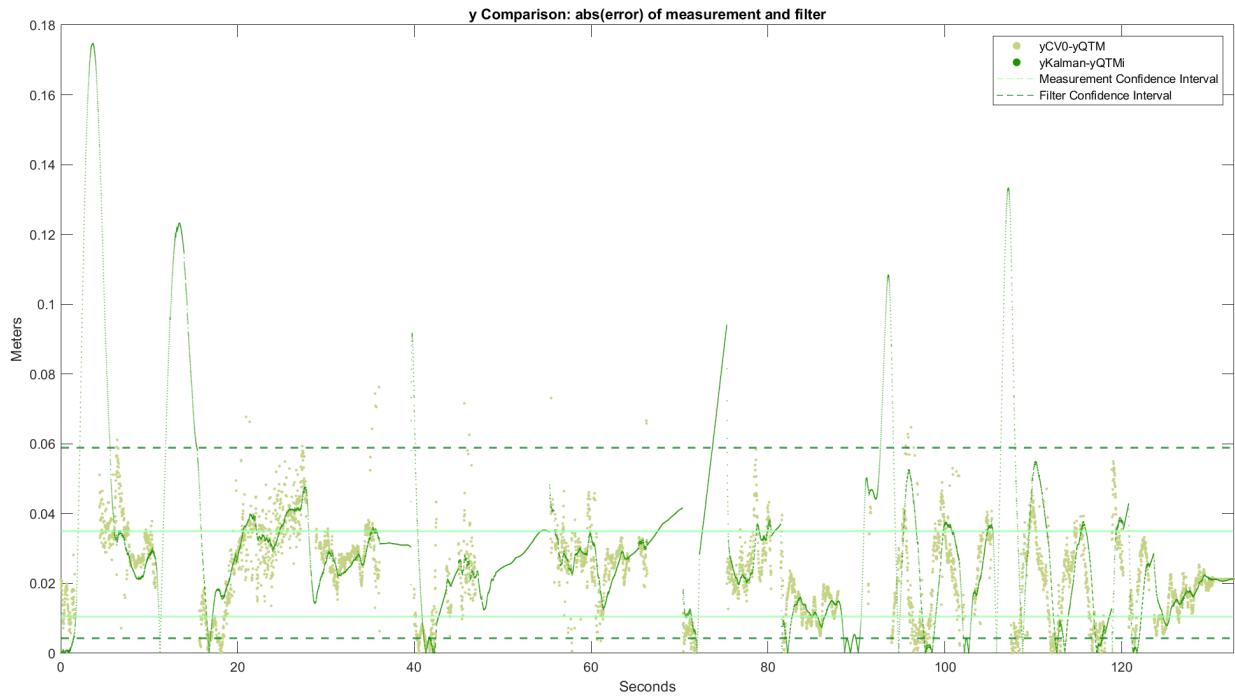


Figure E.76: Confidence intervals for absolute value of y error without and with filter

### E.5.2 Test #1

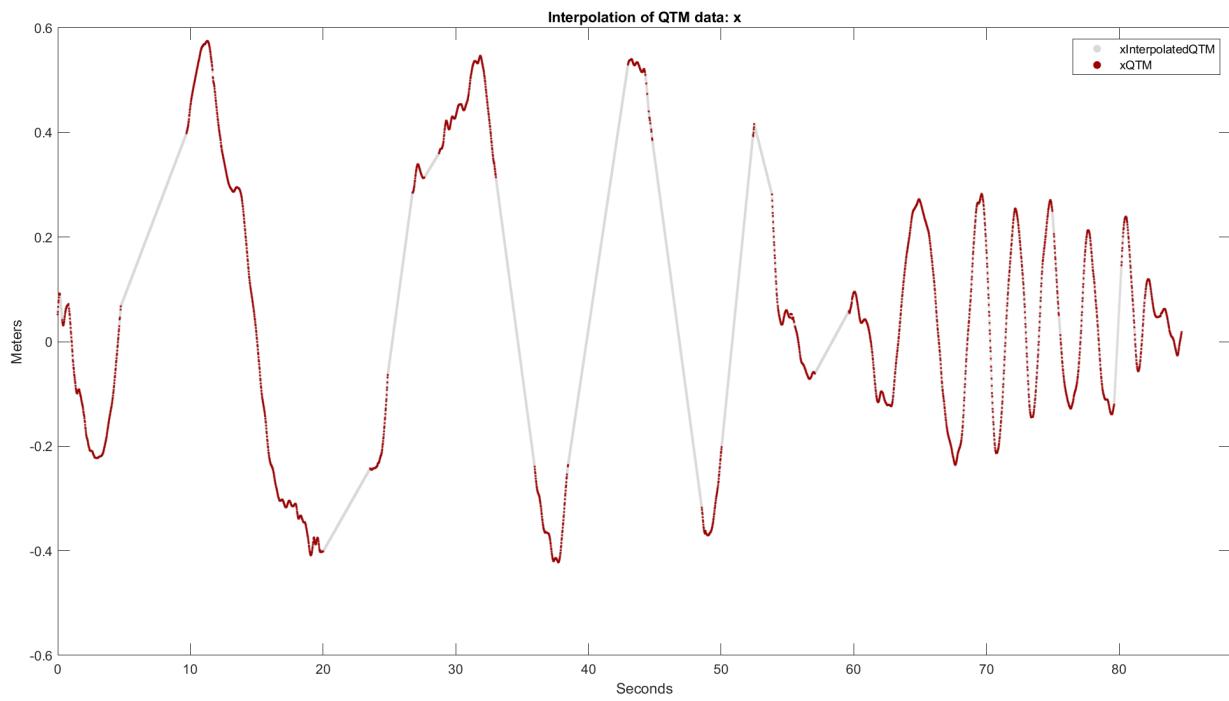


Figure E.77: X interpolation of QTM data

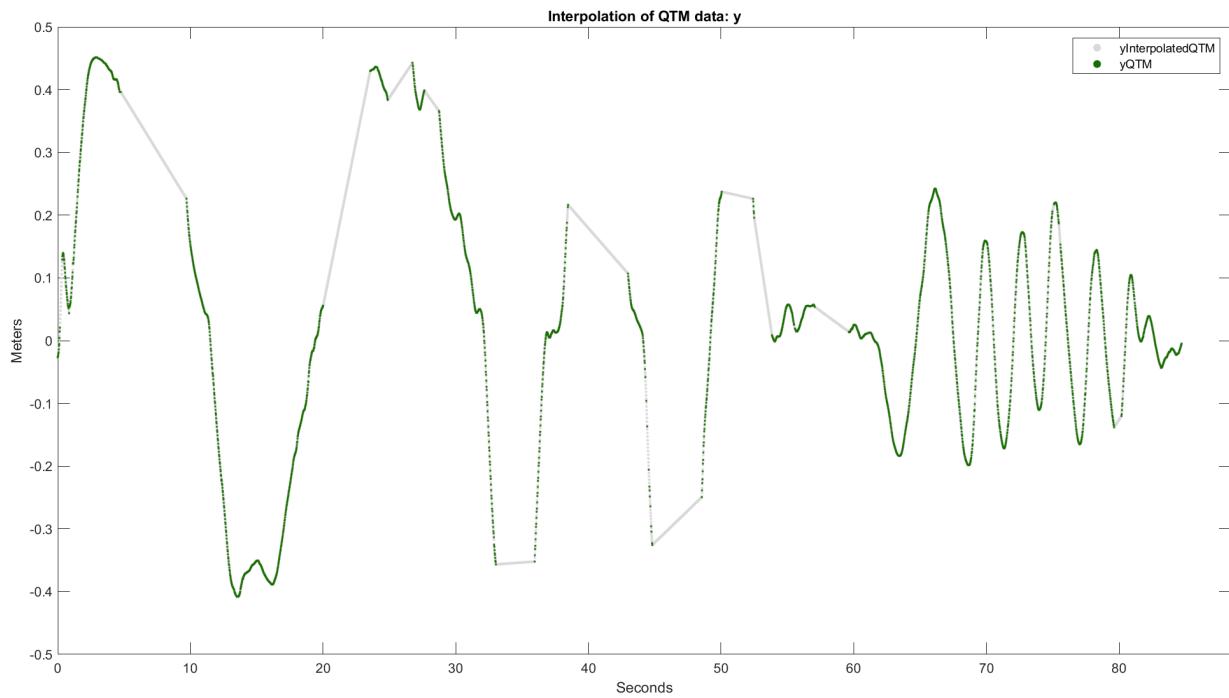


Figure E.78: Y interpolation of QTM data

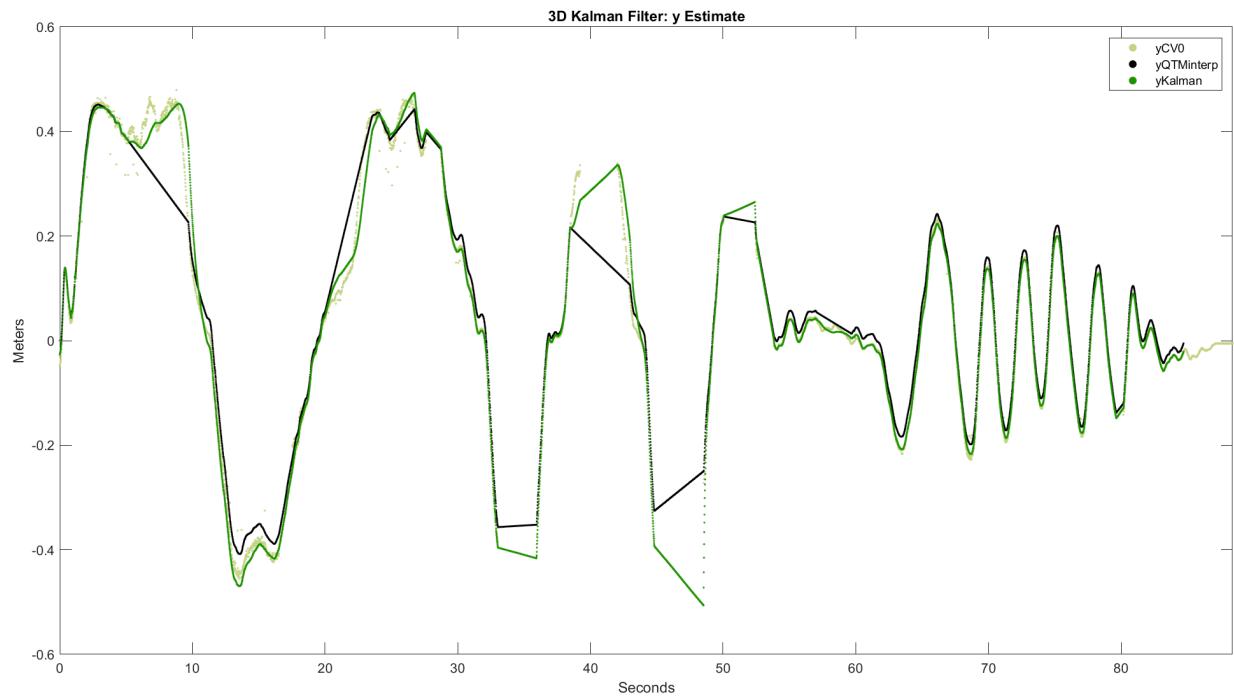


Figure E.79: Comparison of x estimate, measurements and reference

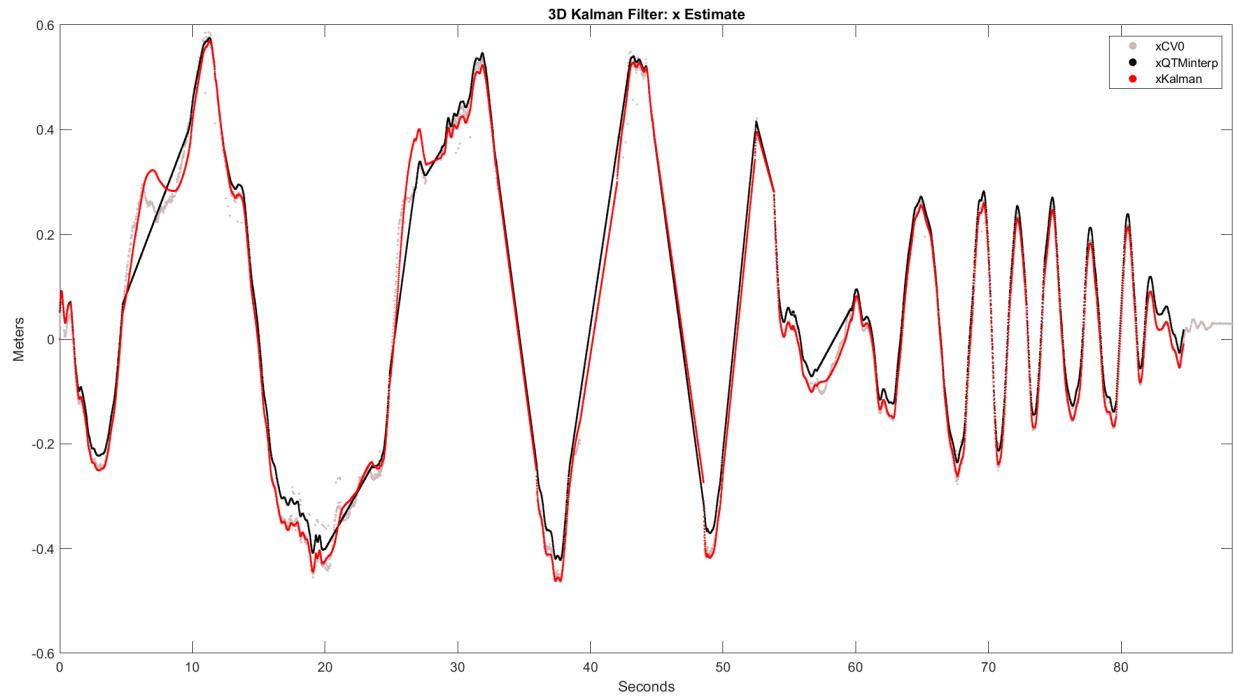


Figure E.80: Comparison of y estimate, measurements and reference

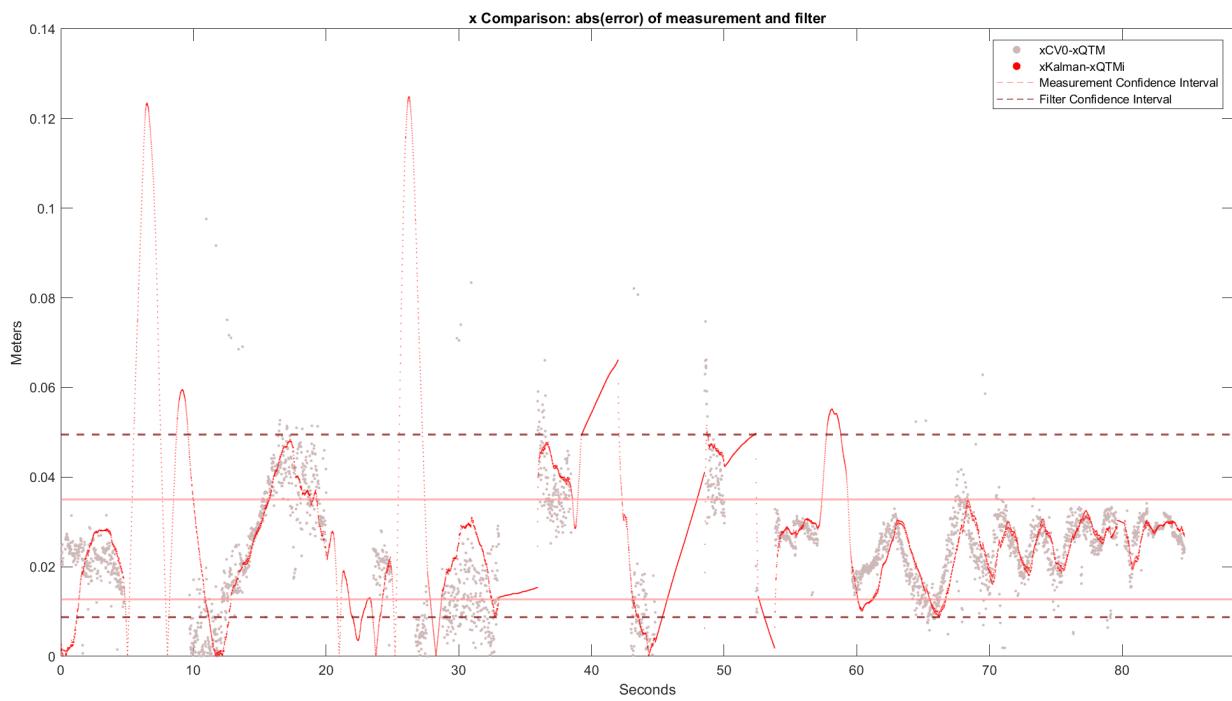


Figure E.81: Confidence intervals for absolute value of x error without and with filter

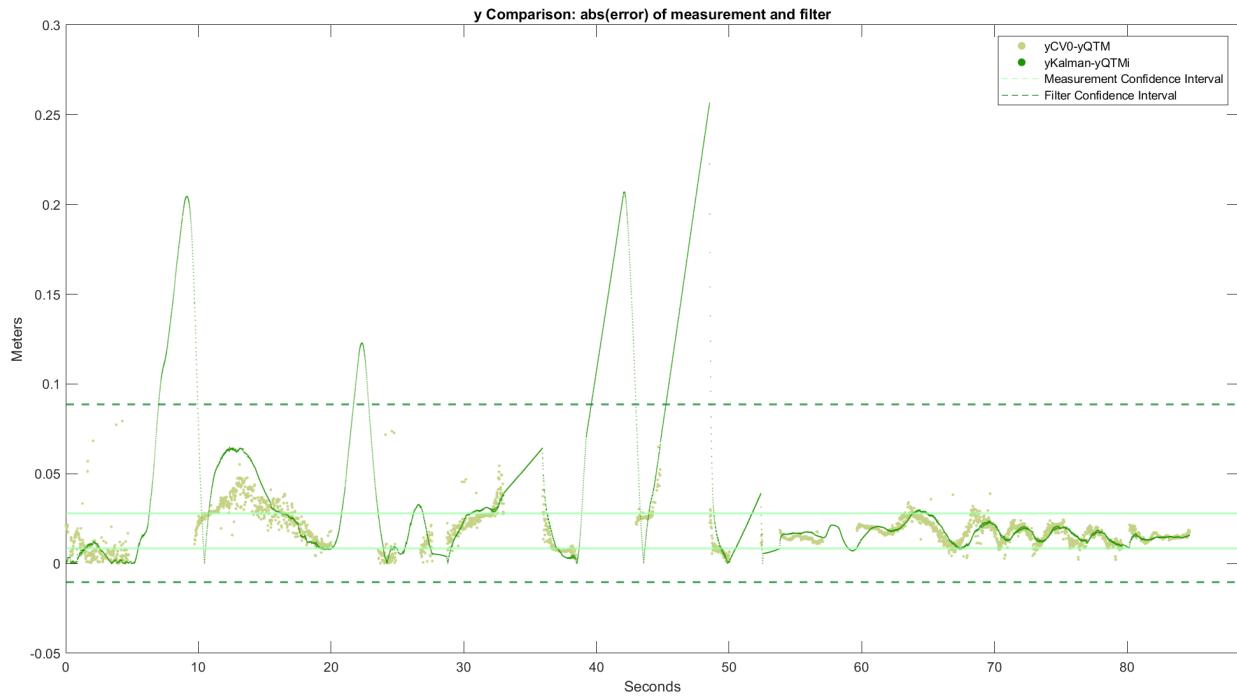


Figure E.82: Confidence intervals for absolute value of y error without and with filter

### E.5.3 Test #2

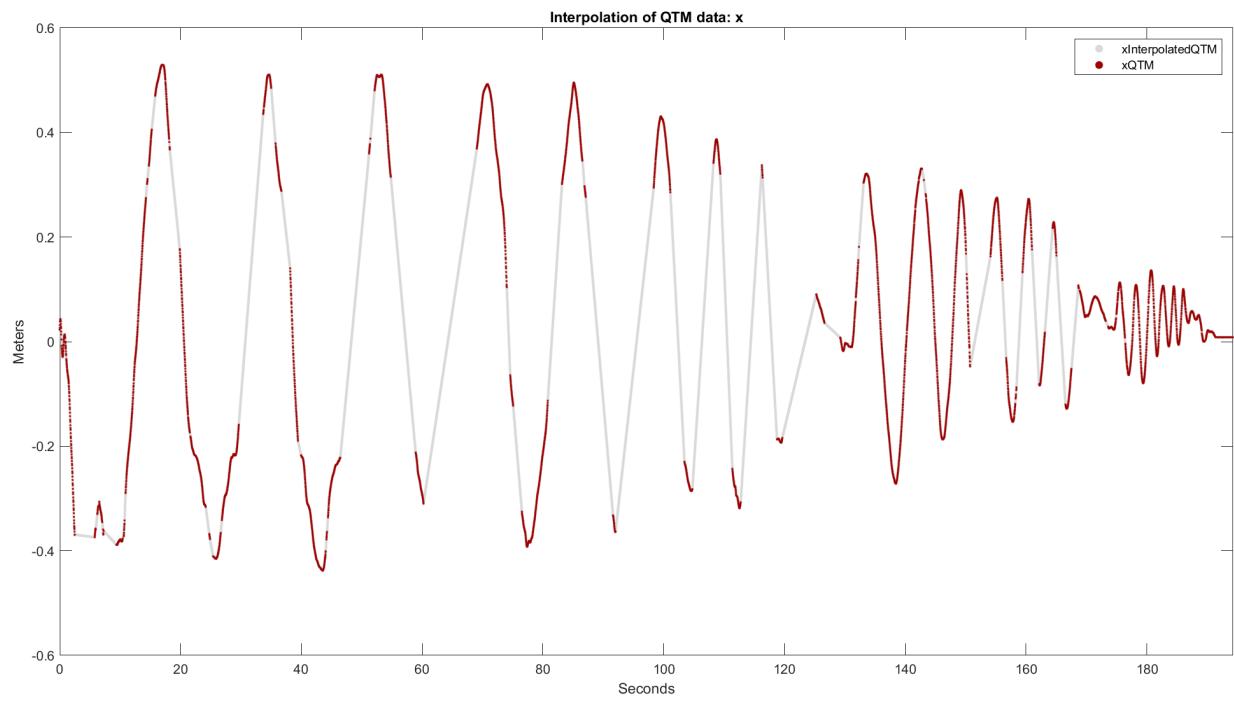


Figure E.83: X interpolation of QTM data

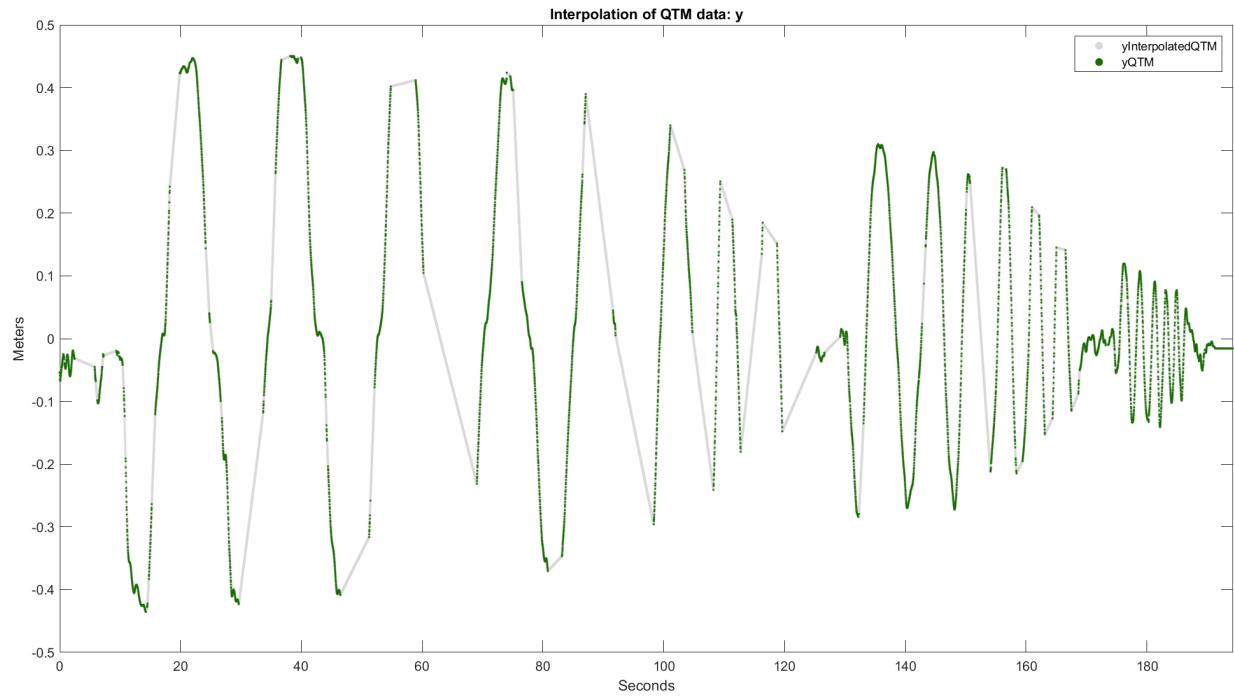


Figure E.84: Y interpolation of QTM data

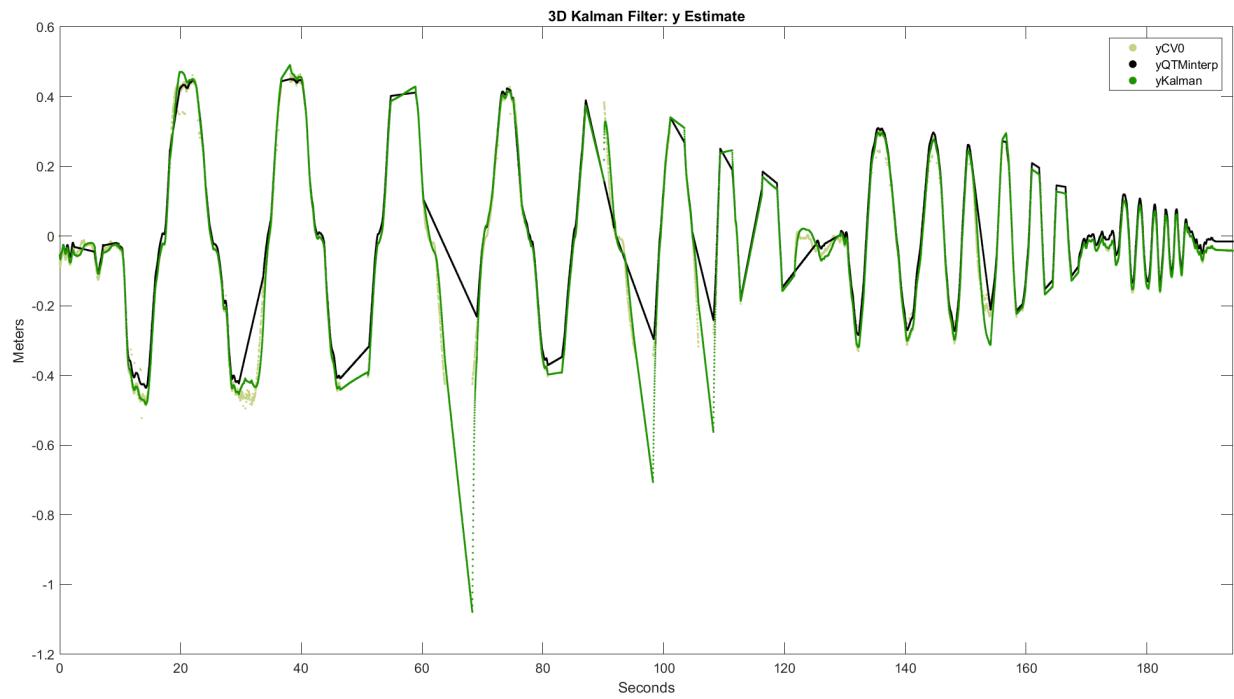


Figure E.85: Comparison of x estimate, measurements and reference

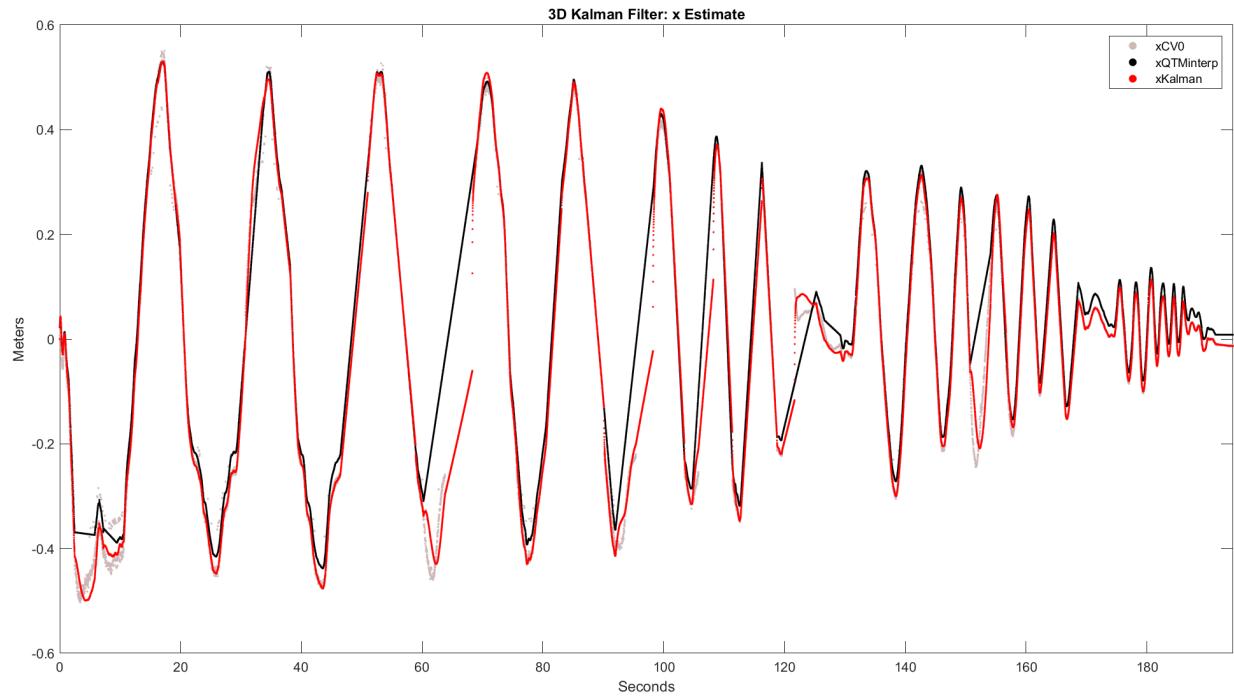


Figure E.86: Comparison of y estimate, measurements and reference

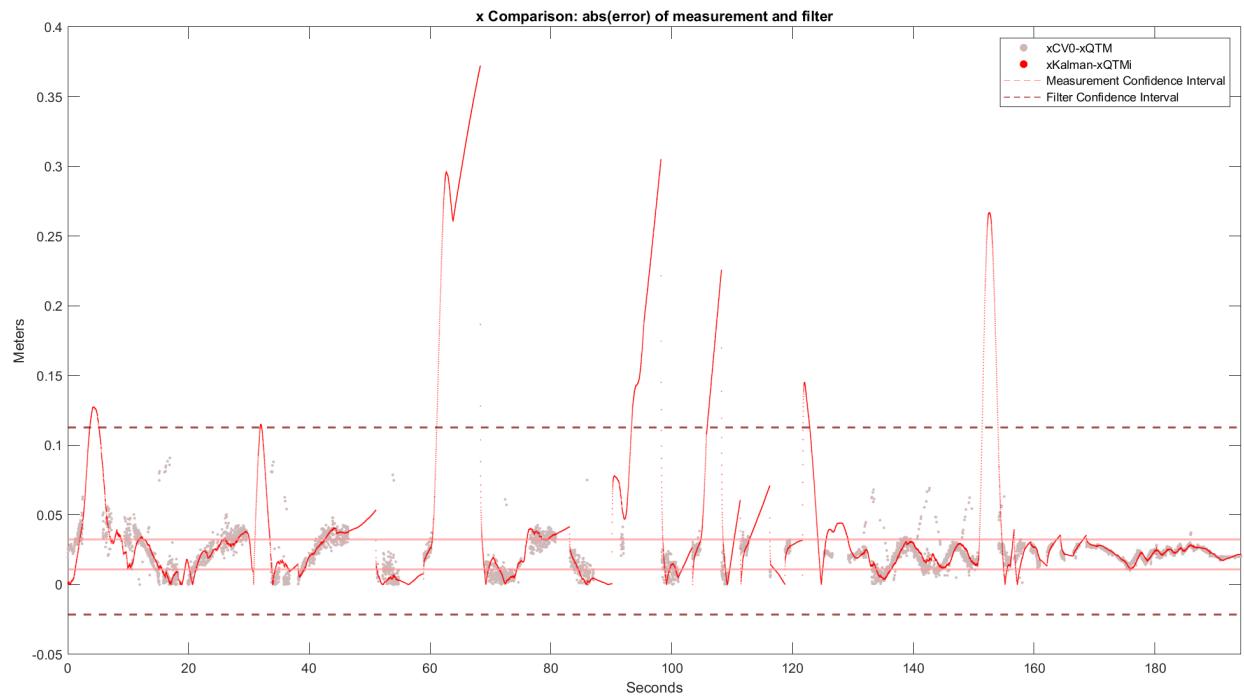


Figure E.87: Confidence intervals for absolute value of x error without and with filter

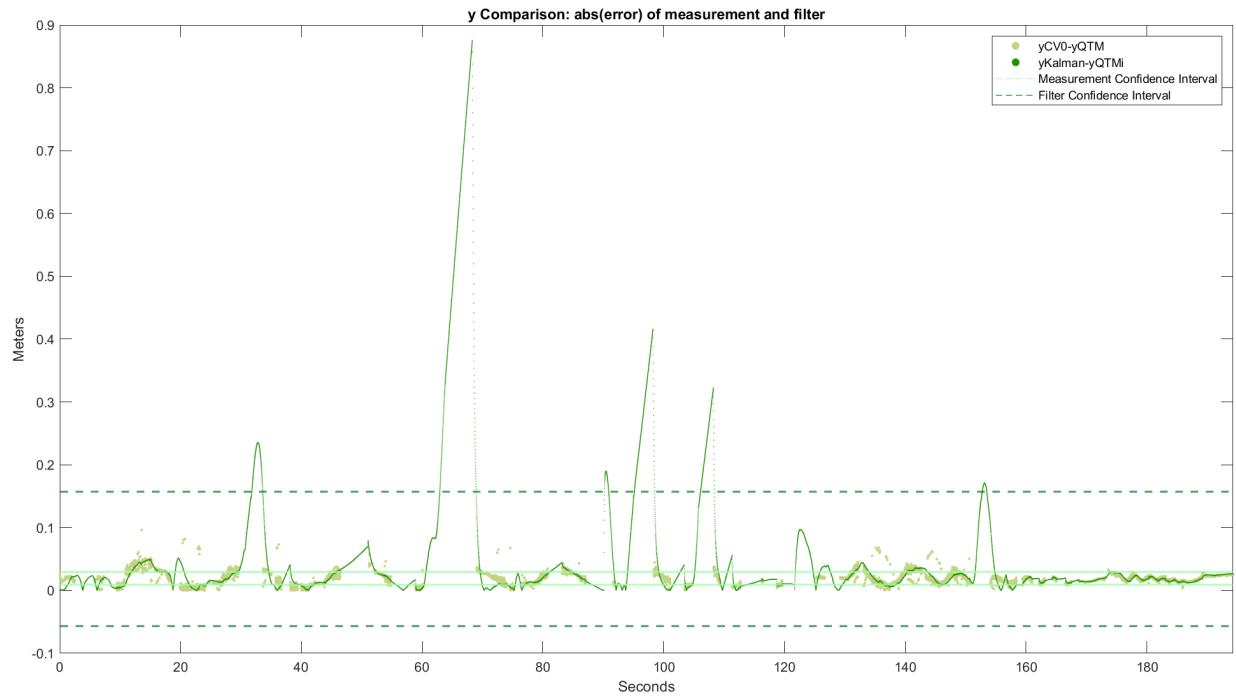


Figure E.88: Confidence intervals for absolute value of y error without and with filter

#### E.5.4 Test #3

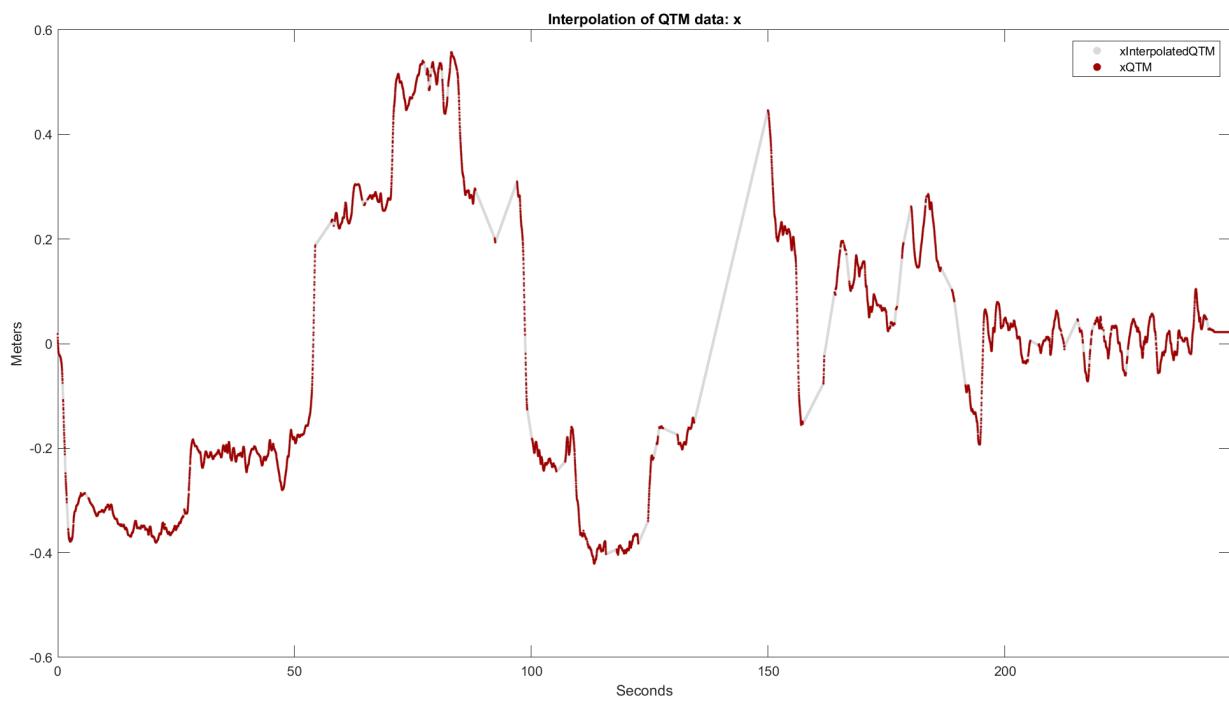


Figure E.89: X interpolation of QTM data

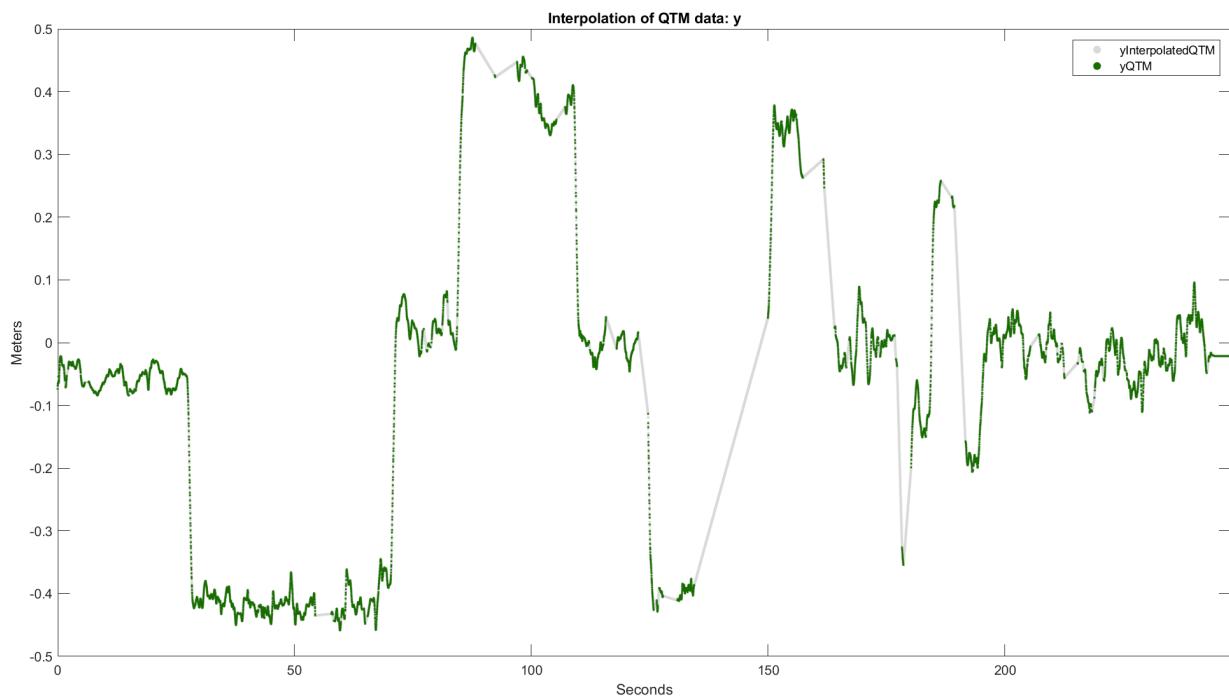


Figure E.90: Y interpolation of QTM data

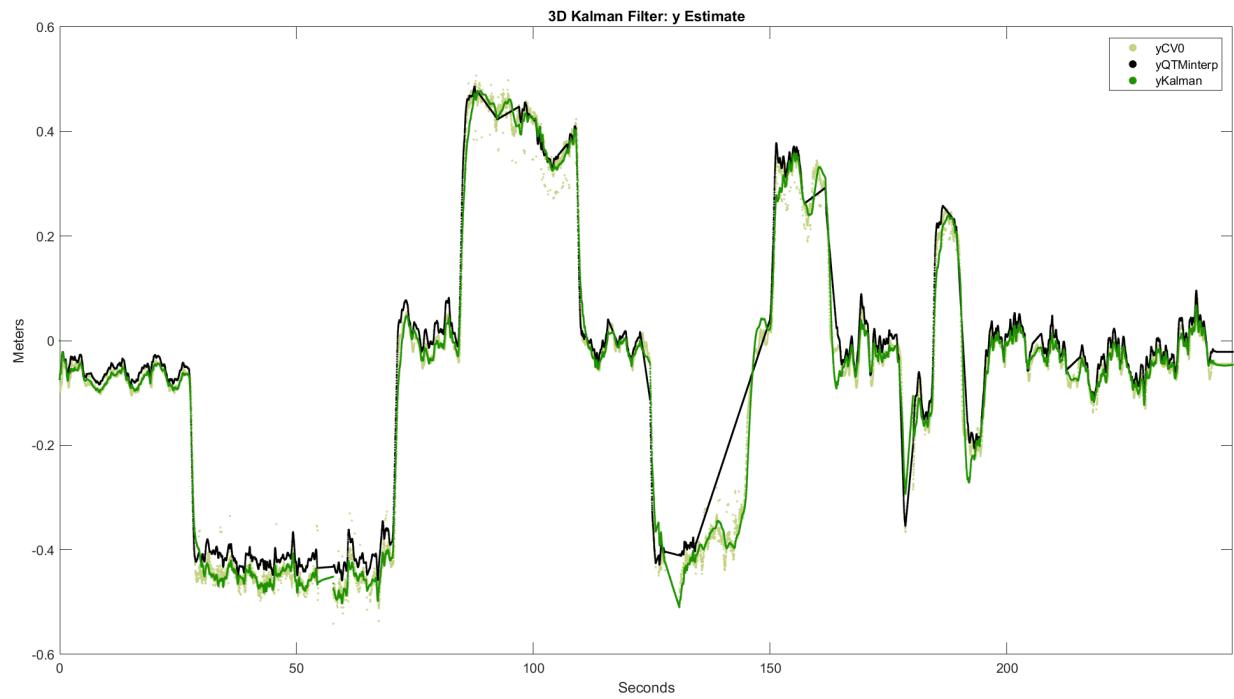


Figure E.91: Comparison of x estimate, measurements and reference

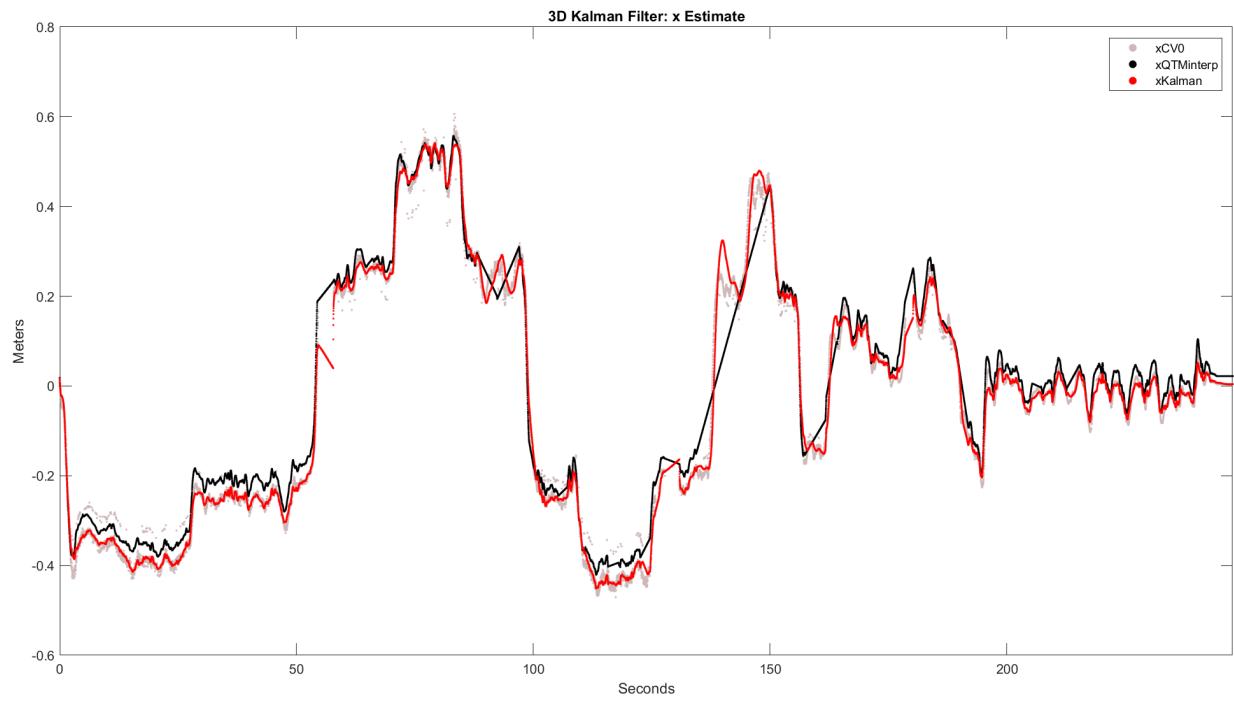


Figure E.92: Comparison of y estimate, measurements and reference

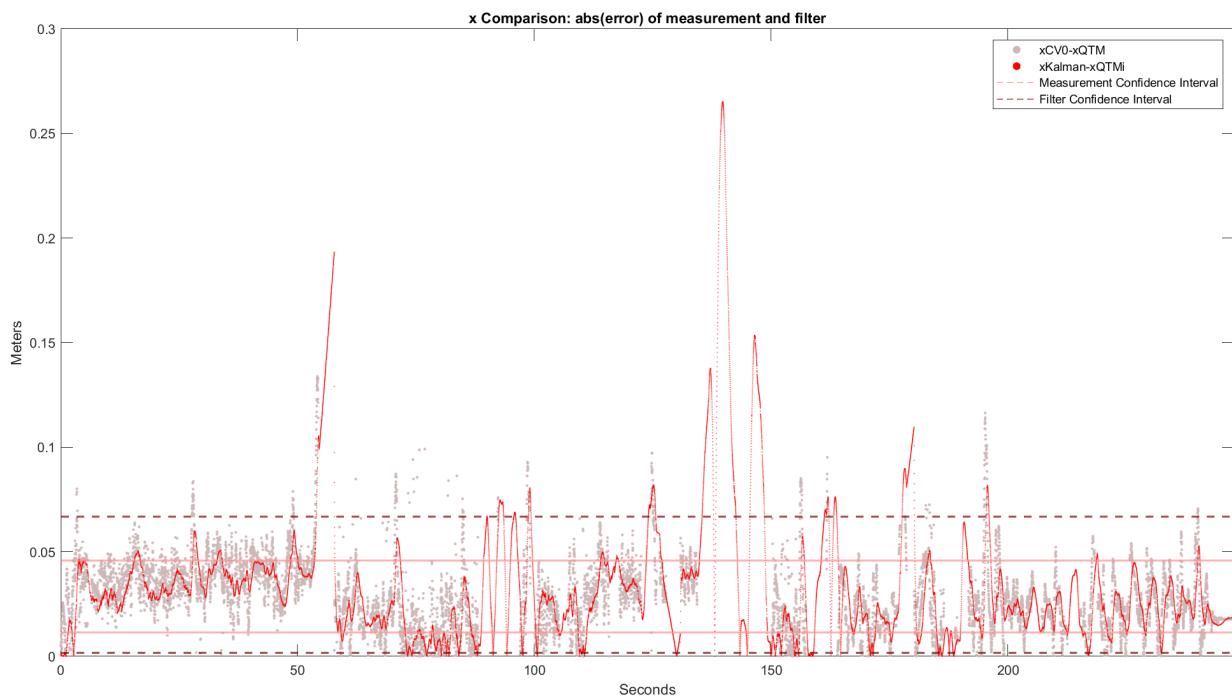


Figure E.93: Confidence intervals for absolute value of x error without and with filter

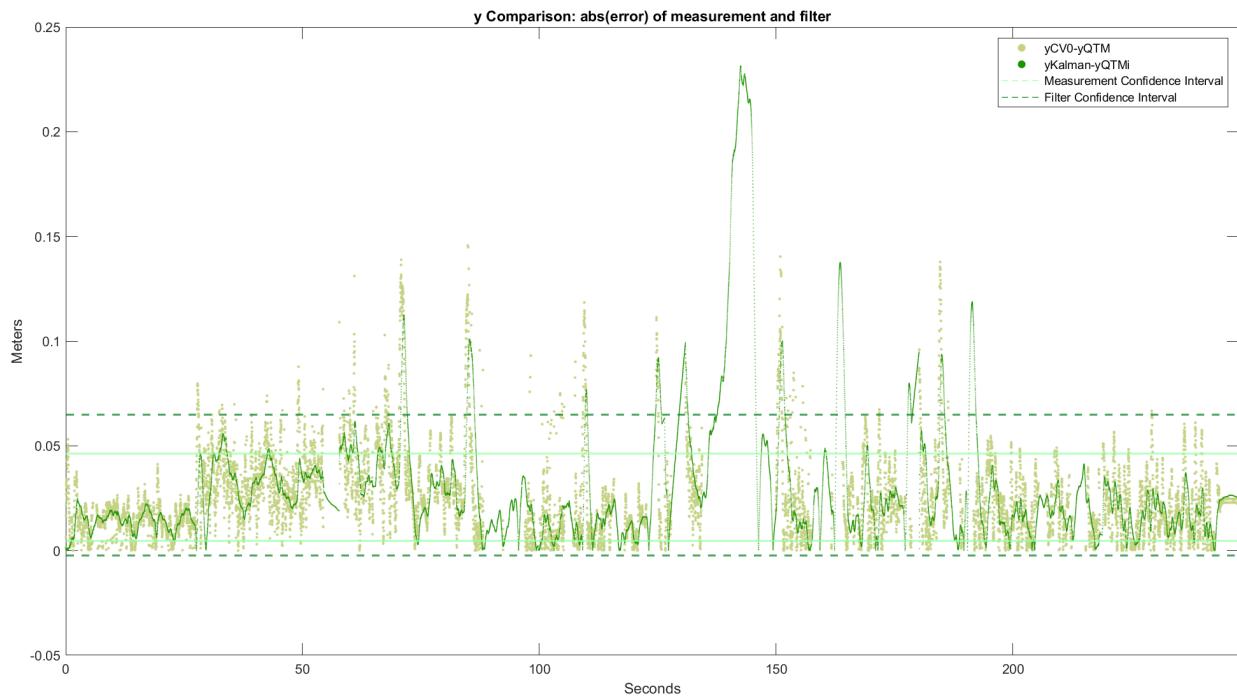


Figure E.94: Confidence intervals for absolute value of y error without and with filter

### E.5.5 Test #4

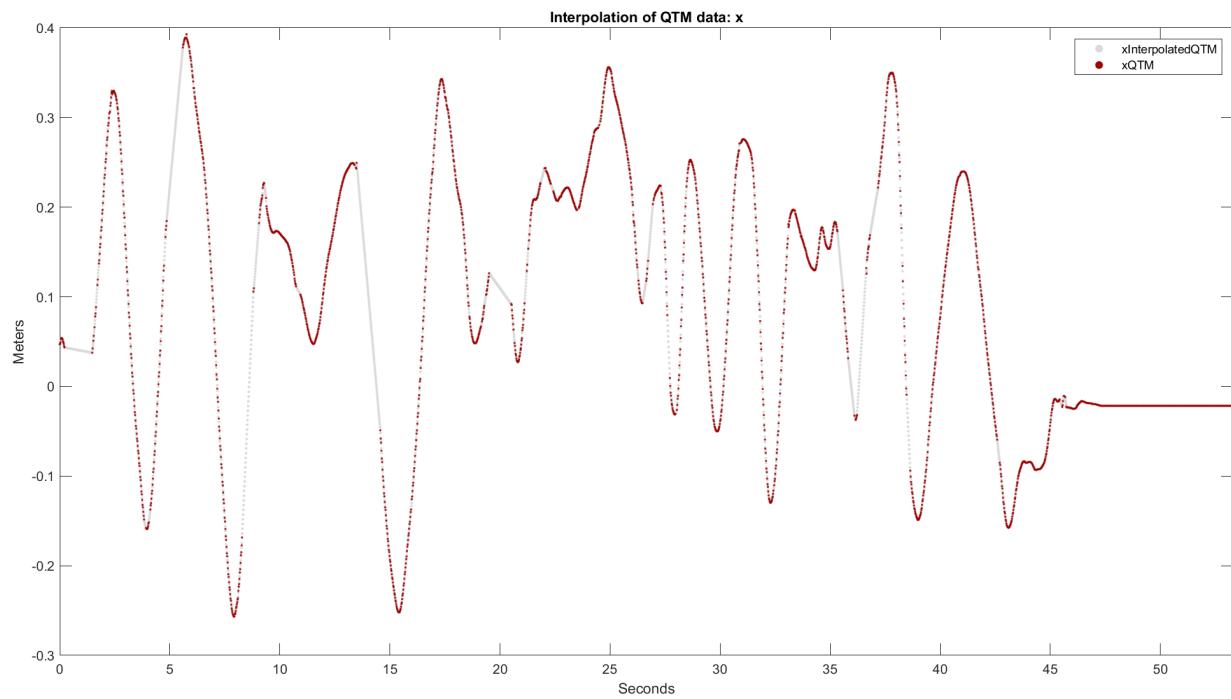


Figure E.95: X interpolation of QTM data

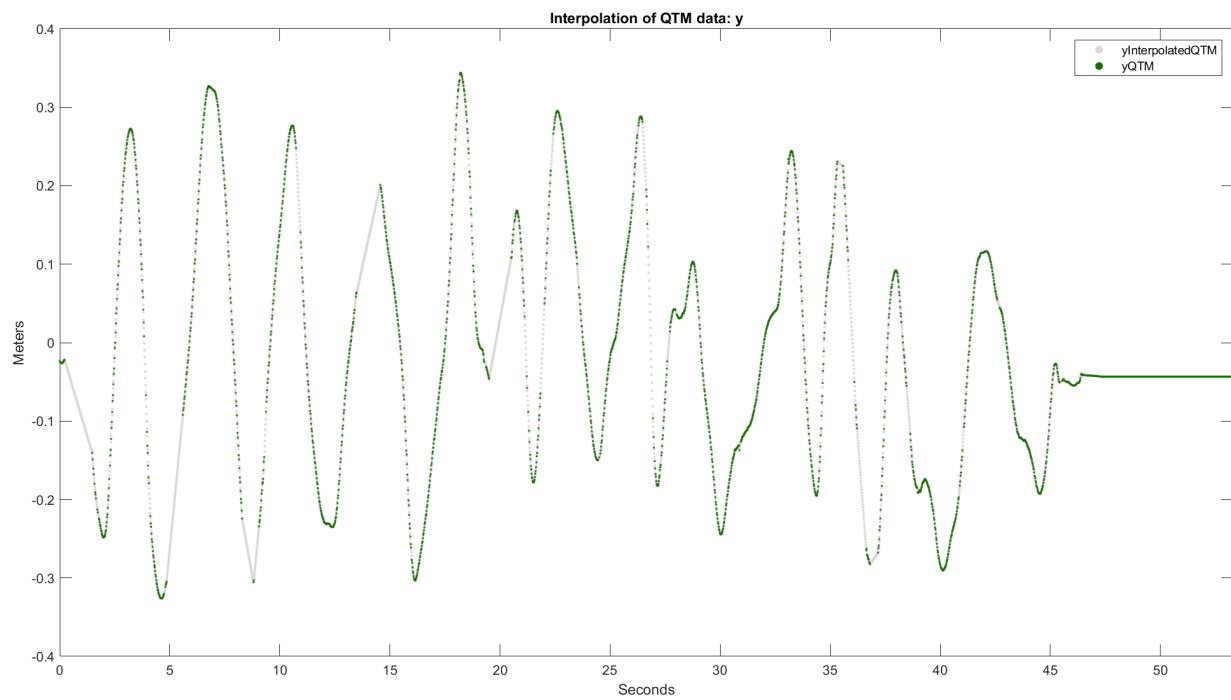


Figure E.96: Y interpolation of QTM data

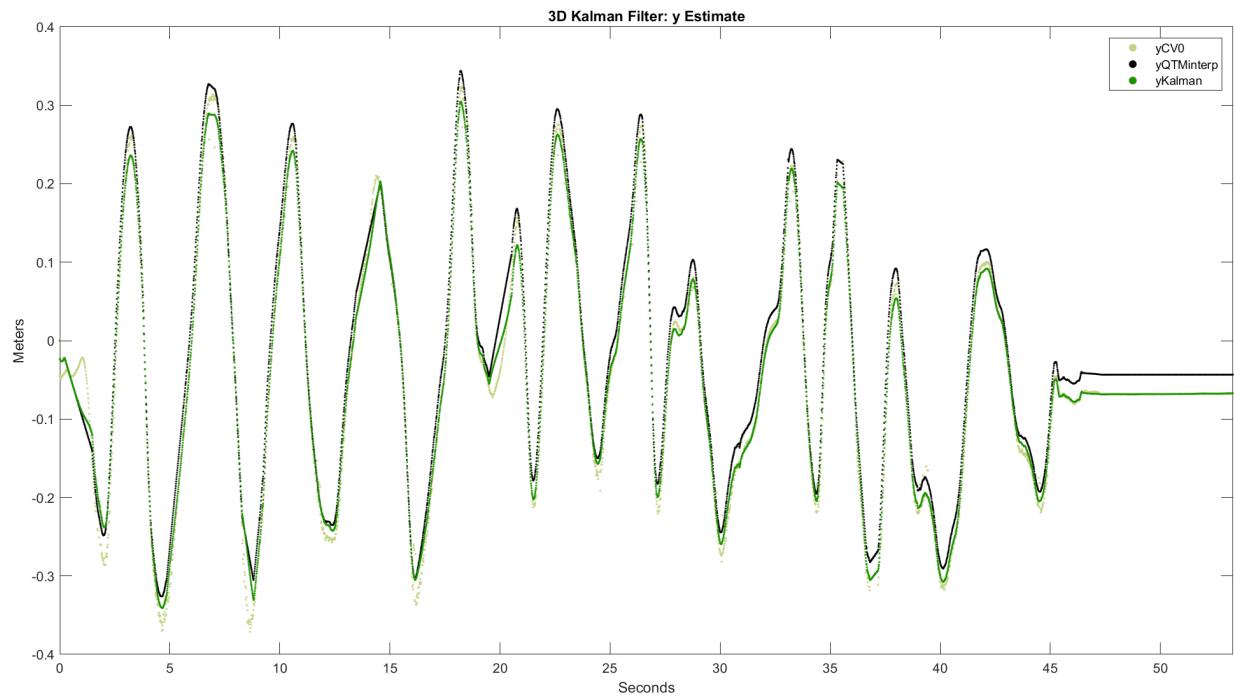


Figure E.97: Comparison of x estimate, measurements and reference

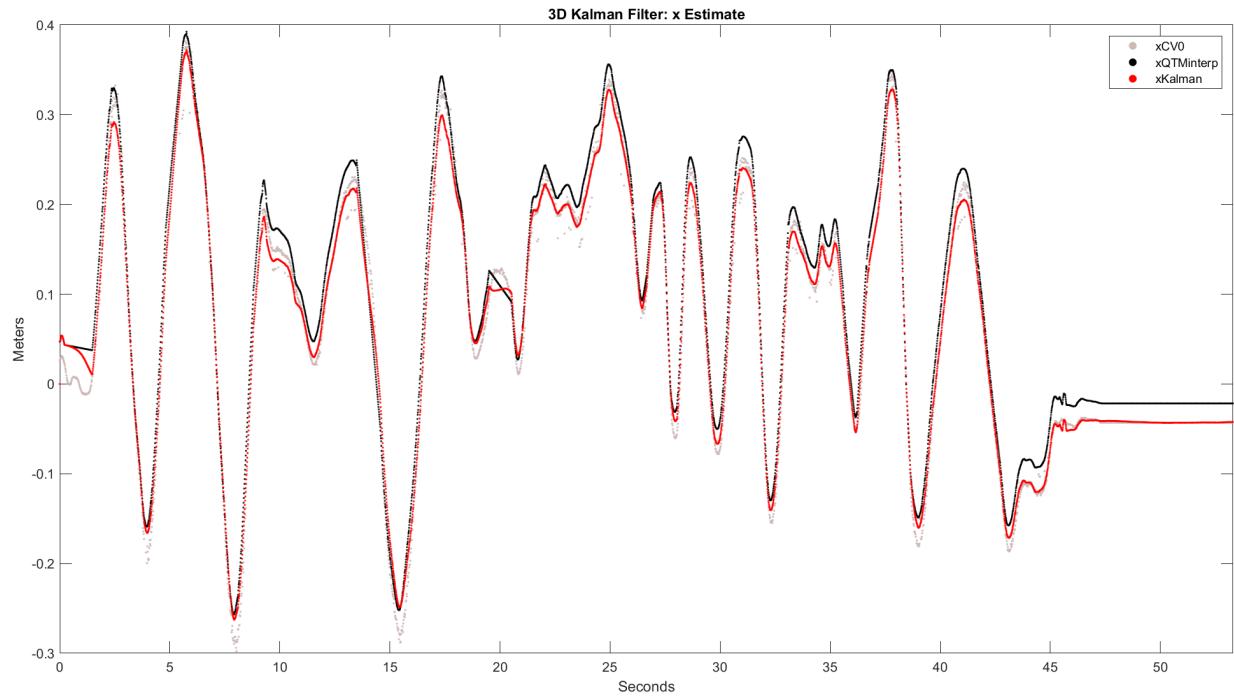


Figure E.98: Comparison of y estimate, measurements and reference

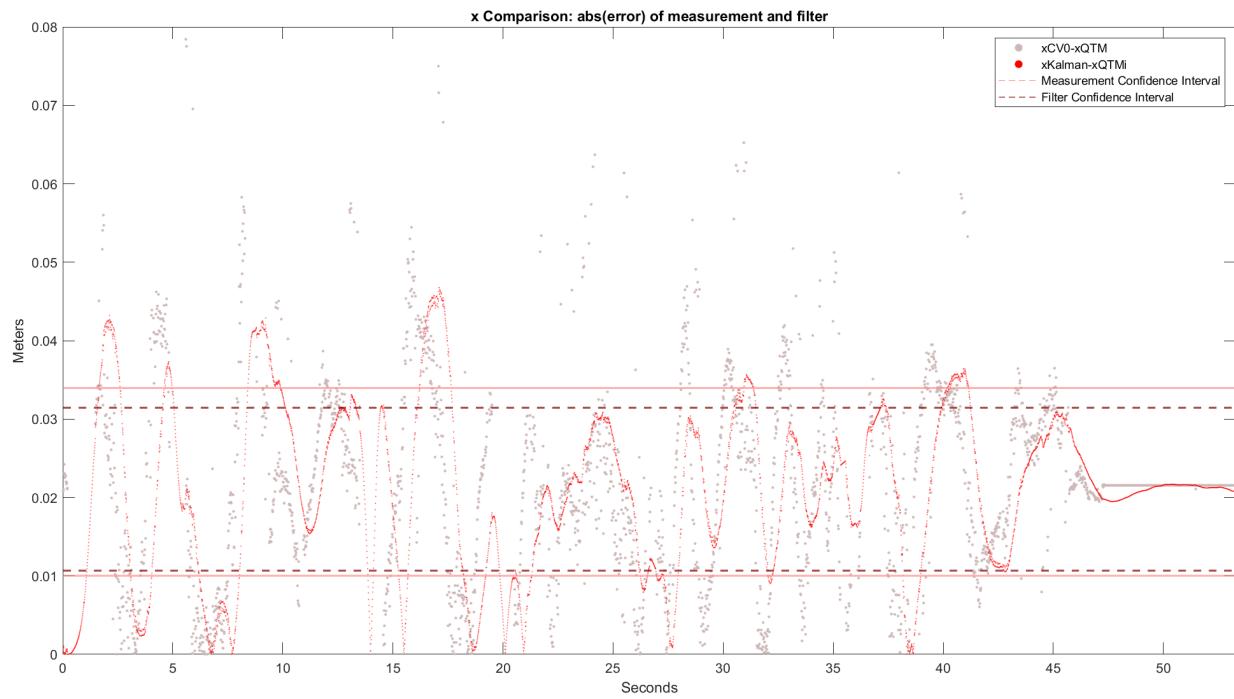


Figure E.99: Confidence intervals for absolute value of x error without and with filter

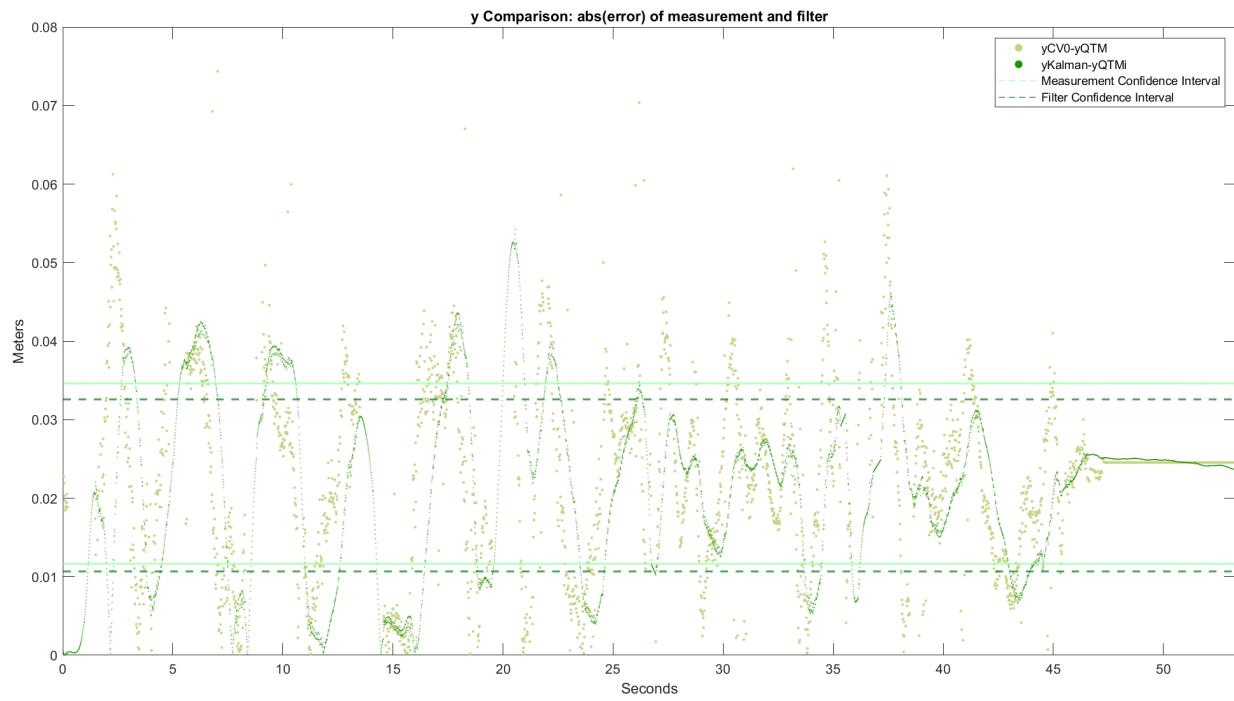


Figure E.100: Confidence intervals for absolute value of y error without and with filter

# F Buildroot

This appendix serves the purpose of a step-by-step tutorial for the buildroot procedure. The language model ChatGPT was utilized for navigating the buildroot procedure, then verifying the suggestions through secondary research.

## F.1 Buildroot OS for RPi CM4

Buildroot 2024.02.2 was downloaded and extracted. A terminal window was opened inside the directory and the command

```
$ make list-defconfigs
```

was run. From the list produced the appropriate config was found under `raspberrypicm4io_64_defconfig`

Subsequently, the command

```
$ make defconfig raspberrypicm4io_64_defconfig
```

was run to apply the appropriate settings for the CM4 A72 CPU.

To further customize the operating system

```
$ make menuconfig
```

was run to bring up the packages menu. "Target options", "toolchain" and "Build options" were left to their default values from the defconfig. Under "system configuration" the system hostname and system banner were modified to reflect the operating systems purpose, but this was a purely cosmetic feature. The root password was set to "mas306". Under the Kernel→Linux Kernel tools the GPIO package was selected. Under Hardware Handling the libgpiod and pigpio packages were added to modify GPIO pin modes and.

To enable wifi and ssh the guide from (stefan cottafavi/) was used. Under Target Packages→Hardware Handling→Firmware, brcmfmac-sdio-firmware-rpi was selected, and from the submenu brcmfmac-sdio-firmware-rpi-wifi. "Rpi 4" and "DTB overlays" should be selected by default here as well. To avoid having to use the modprobe command to load the WiFi module, dev management was set to "Dynamic using Devtmpfs + mdev" under the Systems Configuration menu. Finally, IWD and Dropbear were selected under Target Packages→Networking Applications for logging onto the selected WiFi automatically, and enabling SSH connection. ChatGPT 4.0 was asked to assist with ensuring SPI capabilities are enabled. The following actions were recommended: Run

```
$ make linux-menuconfig
```

and enable SPI Master Controller Drivers, BCM2835 SPI controller, and User mode SPI device driver support under Device Drivers→SPI Support. Under Device Drivers→GPIO Support make sure that Device Tree Overlays are enabled. The default config.txt file already uses dtoverlay, so this should be enabled by default. For troubleshooting and verification purposes, run

```
$ make menuconfig
```

again to add Target Packages→Hardware Handling→SPI-Tools to verify SPI functionality with spidev-test.

The operating system is then built by running

```
$ make
```

in the buildroot directory.

To load the image, the CM4 has to be booted in usb mode. The built in rpiboot from buildroot was unsuccessful, so a different software is used. (jeff gearling source) A git repository containing the usbbboot program is cloned.

```
$ git clone --depth=1 https://github.com/raspberrypi/usbbboot
```

Then, from the usbbboot directory run

```
$ cd usbbboot  
$ make
```

After the OS finished building, the CM4 was placed in the IO board. The jumper to disable eMMC boot was attached together with the micro usb.

Once connected and powered up, the command

```
$ sudo ./rpiboot
```

was run to initiate the usb boot sequence.

After a successful boot, rootfs and a 34MB volume appeared in the device window. A list of drives was generated with

```
$ sudo fdisk -l
```

The CM4 disk name was found to be /dev/sdc. Navigating into the Buildroot directory and running the command

```
$ sudo dd if=output/images/sdcard.img of=/dev/sdc bs=4M
```

flashed the CM4 memory with the new operating system.

After flashing, USB was enabled by entering the 34MB volume and opening the config.txt file and adding the line `otg_mode=1`. Even though USB 2.0 was already enabled with the `dtoverlay` command earlier in the config file, keyboards would not receive power without setting `otg_mode=1`. Additionally to enable SPI the line `dtoverlay=spi0-1cs` was added at the end of the file, as suggested by ChatGPT. This enables SPI channel 0 on chip 1. The config.txt file was then saved and closed.

To enable automatic WiFi connection, a terminal window is opened in the rootfs/var/lib/ directory. To cd into the iwd folder, the command

```
$ sudo -s
```

was run to create a root shell. From here it was possible to `cd` into the iwd directory and create a file named after the desired network name.

```
$ touch <network-SSID>.psk
```

The network SSID file is opened with

```
$ vi <network-SSID>.psk
```

and edited to contain

```
[Security]  
Passphrase=<WiFi-Password>
```

before it was saved by pressing escape twice and typing :w. The root shell was closed by pressing CTRL+D. The CM4 then automatically connected to WiFi upon booting.

## F.2 Cross compiling for the CM4 and running test program

Code running on the CM4 had to be cross compiled for 64 bit ARM architecture. First, the pigpio library was cross compiled by navigating to the download directory and opening a terminal window. ChatGPT is asked to provide the command for cross compiling and installing the library:

```
$ make CROSS_PREFIX=aarch64-linux-gnu-  
sudo make install CROSS_PREFIX=aarch64-linux-gnu-
```

After installing, the BMI088 test program was compiled with

```
$ aarch64-linux-gnu-g++ main.cpp -lpigpio -o BMItest
```

The cross compiled executable was sent to the CM4 using SSH

```
$ scp -C . BMItest root@128.39.202.74:/usr/bin
```

Files transferred show up in the /usr/bin directory.

The BMItest program utilizes the pigpio library, which had an instance running on boot. To free up the pigpio module, the command

```
$ sudo killall pigpiod
```

allows BMItest to access the library.

Navigating to /usr/bin/ in SSH on the CM4, BMItest was run using

```
$ ./BMItest
```

To check the RAM and disk space usage of the OS, the linux commands

```
$ free -m  
$ df
```

were used respectively, the results of this for the generated OS can be found in 5.1.

# G Source Code

## G.1 Machine Vision Development

### G.1.1 depthCameraTest.py

```
# Copied from Nick DiFilippo:  
# https://github.com/nickredsox/youtube/blob/master/Robotics/realsense.py  
  
import pyrealsense2 as rs  
import numpy as np  
import cv2  
  
pipe = rs.pipeline()  
cfg = rs.config()  
  
cfg.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)  
cfg.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)  
  
pipe.start(cfg)  
  
while True:  
    frame = pipe.wait_for_frames()  
    depth_frame = frame.get_depth_frame()  
    color_frame = frame.get_color_frame()  
  
    depth_image = np.asanyarray(depth_frame.get_data())  
    color_image = np.asanyarray(color_frame.get_data())  
    depth_cm = cv2.applyColorMap(cv2.convertScaleAbs(depth_image,  
                                                    alpha = 0.5), cv2.COLORMAP_JET)  
  
    gray_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)  
  
    cv2.imshow('rgb', color_image)  
    cv2.imshow('depth', depth_cm)  
  
    if cv2.waitKey(1) == ord('q'):  
        break  
  
pipe.stop()
```

### G.1.2 screenshotCamera.py

```
import cv2           # OpenCV
import pyrealsense2 as rs   # RealSense wrapper
import numpy          # Python math
import os             # For path

# Depth Camera connection
pipe = rs.pipeline()
config = rs.config()

config.enable_stream(rs.stream.color, 848, 480, rs.format.bgr8, 60) # ...
    (streamType, xRes, yRes, format, fps)

pipe.start(config)

# Frame counter
frameNr = 0

# Get the current working directory
curDir = os.getcwd()

# Screenshot directory
screenDir = os.path.join(curDir, 'Screenshots')

# Calibration screenshots directory
calibDir = os.path.join(curDir, 'calibrationCaps')

while(True):

    # Collect frames from camera (depth, color, IR)
    frame = pipe.wait_for_frames()

    # Toggle these three lines when changing config from (depth,color,IR)
    #frame = frame.get_infrared_frame()
    frame = frame.get_color_frame()
    #frame = frame.get_depth_frame()

    # Convert to numpy array
    image = numpy.asarray(frame.get_data())

    # Add color coding if using depth stream
    #image = cv2.applyColorMap(cv2.convertScaleAbs(image, alpha=1), ...
        cv2.COLORMAP_TURBO)

    # Display the current frame
    cv2.imshow('LiveReading', image)

    # Store key pressed during 1 [ms] delay
    keyPressed = cv2.waitKey(1)
```

```

# S to take screenshot
if keyPressed == ord('s'):
    os.chdir(screenDir) # Change directory
    # ----- ChatGPT -----
    # Get the list of items (files and folders) inside the folder
    items = os.listdir(screenDir)
    # Count the number of items
    num_items = len(items)
    # ----- ChatGPT -----
    cv2.imwrite(filename=f"Screenshot_{num_items}.jpg", img=image) # ...
        Incrementing filename
    print('Screenshot successful!')

# C to capture for calibration
if keyPressed == ord('c'):
    os.chdir(calibDir) # Change this for screenshots or for calibrationr)
    cv2.imwrite(filename=f"Screenshot_frame{frameNr}.jpg", img=image) ...
        # Incrementing filename
    print('Calibration frame captured!')

# Q to stop stream
elif keyPressed == ord('q'):
    break
frameNr += 1

pipe.stop()          # Stop recording
cv2.destroyAllWindows() # Free resources

```

### G.1.3 arucoDetection.py

```
# ----- Libraries -----
import cv2          # OpenCV
import cv2.aruco as aruco # For simplification
import pyrealsense2 as rs # RealSense Wrapper
import numpy         # Python Math
import os            # Path Control
# ----- Libraries -----

# ----- Constant variables for simple changes -----
markerColor = (255, 255, 0) # Corner color is set to be contrast to border ...
color

font = cv2.FONT_HERSHEY_SIMPLEX
fontColor = (200, 20, 200)
fontScale = 0.5
fontThickness = 1
# ----- Constant variables for simple changes -----

# ----- Save Frame -----
# Directory to save image
dir = r'/home/thomaz/MAS306_Drone_G12/Code/Camera/Screenshots'
os.chdir(dir)

# ----- ChatGPT -----
# Get the list of items (files and folders) inside the folder
items = os.listdir(dir)
# Count the number of items
num_items = len(items)
# ----- ChatGPT -----
# ----- Save Frame -----

# Set dictionary for the markers
arucoDictionary = aruco.getPredefinedDictionary(aruco.DICT_6X6_50)

# Setup configuration and start pipeline stream
pipe = rs.pipeline()
config = rs.config()

config.enable_stream(rs.stream.color, 848, 480, rs.format.bgr8, 60) # ...
(streamType, xRes, yRes, format, fps)
pipe.start(config)

while(True):

    # Frame Collection
    frame = pipe.wait_for_frames()           # collect frames from camera ...
    (depth, color, etc)
    color_frame = frame.get_color_frame()     # Extract RGB module frame
    color_image = numpy.asarray(color_frame.get_data()) # Convert to ...
    NumPy array
```

```

# Identification
gray = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)      # Grayscale image
corners, ids, rejectedImagePoints = aruco.detectMarkers(gray, ...
    arucoDictionary, parameters=aruco.DetectorParameters())

# Information display in terminal
print(f'\n[INFO] ArUco Marker ID: {ids}') # Array with current ...
IDs: [ id1 ] [id2] ... [idn]
print('[INFO] Corners: ')
print(corners) # Matrix with current corner coordinates [ x1, y1; x2, ...
    y2; x3, y3; x4, y4 ], [-|-], ...

# Display outline of markers
color_image = aruco.drawDetectedMarkers(color_image, corners, ...
    borderColor=markerColor)

# ----- v Display ID of marker, by pyImageSearch v -----
if len(corners) > 0: # So iteration makes sense

    ids = ids.flatten() # Remove square brackets for text

    for (markerCorner, markerID) in zip(corners, ids): # alias on ...
        left, so can iterate in parallel
        # Extract corners
        corners = markerCorner.reshape(4,2)
        (topLeft, topRight, bottomRight, bottomLeft) = corners

        # convert each of the (x, y)-coordinate pairs to integers
        topRight = (int(topRight[0]), int(topRight[1]))
        bottomRight = (int(bottomRight[0]), int(bottomRight[1]))
        bottomLeft = (int(bottomLeft[0]), int(bottomLeft[1]))
        topLeft = (int(topLeft[0]), int(topLeft[1]))

        IDtext = f'ID: {markerID}'

        # Display on the image
        cv2.putText(color_image, IDtext,(topLeft[0], topLeft[1] - 15), ...
            font, fontScale, fontColor, fontThickness)
# ----- ^ Display ID of marker, by pyImageSearch ^ -----


# Display image
cv2.imshow('LiveReading', color_image)

# Store key pressed during 1 [ms] delay
keyPressed = cv2.waitKey(1)

# Check for stored key
if keyPressed == ord('s'): # S to save frame
    cv2.imwrite(filename=f"screenshot_{num_items}.jpg", ...
        img=color_image) # Incrementing filename
    print('Screenshot successful!')
elif keyPressed == ord('q'): # Q to stop stream
    break

pipe.stop() # Stop recording
cv2.destroyAllWindows() # Free resources

```

#### G.1.4 arucoPoseEstimation.py

```
# ----- Libraries -----
import cv2
import cv2.aruco as aruco # For simplification
import pyrealsense2 as rs
import numpy
# ----- Libraries -----

# ----- Constant variables for simple changes -----
#markerColor = (255, 255, 0) # Corner color is set to be contrast to ...
# border color

font = cv2.FONT_HERSHEY_SIMPLEX
fontColor = (200, 20, 200) # bgr
fontScale = 0.5
fontThickness = 1
axesLength = 0.01
screenHeight = 480 # pixels
screenWidth = 848 # pixels
cColor = (0, 0, 120) # bgr
cThick = 1 # pixels

markerSize = 0.05 # Length of ArUco marker sides [m]

# ----- ALL INTRINSICS ARE FOR 848x480 -----

# From Software Development Kit
# cameraMatrix = numpy.array([
#     [613.037048339844, 0, 429.841949462891], # [f_x, ...
#     0.0, c_x] used principal points
#     [0, 612.738342285156, 237.866897583008], # [0.0, ...
#     f_y, c_y] as optical center points
#     [0, 0, 1.0] ]) # [0.0, ...
#     0.0, 1.0]
distortionCoefficients = numpy.array(
    [0.0, 0.0, 0.0, 0.0, 0.0]) # [k1, k2, p1, p2, k3]

# Calibration v2 opencv
cameraMatrix = numpy.array([
[608.76301751, 0.0, 429.37397121],
[0.0, 609.23981796, 232.71315263],
[0.0, 0.0, 1.0]]])
# distortionCoefficients = numpy.array(
#     [0.21043186, -0.69360305, -0.00180299, 0.00226683, 0.65082012]) # ...
# [k1, k2, p1, p2, k3]

print("\nCamera Matrix\n", cameraMatrix)
print("\nDistortion Coefficients\n", distortionCoefficients)

# ----- Constant variables for simple changes -----

# Set dictionary for the markers
arucoParams = aruco.DetectorParameters()
arucoDictionary = aruco.getPredefinedDictionary(aruco.DICT_ARUCO_ORIGINAL)

# Setup configuration and start pipeline stream
pipe = rs.pipeline()
config = rs.config()
```

```

config.enable_stream(rs.stream.color, screenWidth, screenHeight, ...
    rs.format.bgr8, 60) # (streamType, xRes, yRes, format, fps)
pipe.start(config)

transVector = [0, 0, 0]
while(True):

    # Frame Collection
    frame = pipe.wait_for_frames()                      # collect frames from camera ...
    (depth, color, etc)
    color_frame = frame.get_color_frame()    # Extract RGB module frame
    color_image = numpy.asarray(color_frame.get_data()) # Convert to ...
                                                NumPy array

    # Identification
    gray = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)    # Grayscale image
    corners, ids, rejectedImagePoints = aruco.detectMarkers(gray, ...
        arucoDictionary, parameters=arucoParams)

    # ----- Pose Estimation -----
    if len(corners) > 0:
        for i in range(0, len(ids)):

            rotVector, transVector, markerPoints = ...
                aruco.estimatePoseSingleMarkers(
                    corners[i], markerSize, cameraMatrix=cameraMatrix, ...
                    distCoeffs=distortionCoefficients)

            cv2.drawFrameAxes(color_image, cameraMatrix=cameraMatrix,
                distCoeffs=distortionCoefficients, ...
                rvec=rotVector, tvec=transVector, ...
                length=axesLength)

            transVector = numpy.around(transVector, 4)

            print("\nTranslation Vector: ", transVector)
    # ----- Pose Estimation -----


    # Information display in terminal
    # print(f'\n[INFO] ArUco Marker ID:      {ids}') # Array with current ...
    #     IDs: [ [id1] [id2] ... [idn] ]
    # print('[INFO] Corners: ')
    # print(corners) # Matrix with current corner coordinates [ x1, y1; ...
    #     x2, y2; x3, y3; x4, y4 ], [-|-|], ...


    # ----- v Display ID of marker, by pyImageSearch v -----
    ids = ids.flatten() # Remove square brackets for text

    for (markerCorner, markerID) in zip(corners, ids): # alias on ...
        left, so can iterate in parallel
        # Extract corners
        corners = markerCorner.reshape(4,2)
        (topLeft, topRight, bottomRight, bottomLeft) = corners

```

```

# convert each of the (x, y)-coordinate pairs to integers
topRight      = (int(topRight[0]), int(topRight[1]))
bottomRight   = (int(bottomRight[0]), int(bottomRight[1]))
bottomLeft    = (int(bottomLeft[0]), int(bottomLeft[1]))
topLeft       = (int(topLeft[0]), int(topLeft[1]))

IDtext = f'ID: {markerID}'

# Display on the image
cv2.putText(color_image, IDtext, (topLeft[0], topLeft[1] - 15), ...
            font, fontScale, fontColor, fontThickness)
# ----- ^ Display ID of marker, by pyImageSearch ^ -------

# Display crosshair horizontal then vertical
color_image = cv2.line(color_image, (0, int(screenHeight/2)), ...
                       (screenWidth, int(screenHeight/2)), cColor, cThick)
color_image = cv2.line(color_image, (int(screenWidth/2), 0), ...
                       (int(screenWidth/2), screenHeight), cColor, cThick)

# Display image
cv2.imshow('LiveReading', color_image)

pressedKey = cv2.waitKey(1)

# Loop breaker
if pressedKey == ord('q'):          # Press Q to stop
    break
elif pressedKey == ord('p'):         # Press P to pause
    cv2.waitKey(-1)

pipe.stop()                         # Stop recording
cv2.destroyAllWindows() # Free resources

```

### G.1.5 cameraCalibration.py

```
# ----- Libraries -----
import cv2          # OpenCV
import numpy        # Python Math
import glob         # For importing images
import os           # For export path

# Don't need to import pyrealsense2 as we are using from screenshotCamera.py
# ----- Libraries -----

# Get the current working directory
curDir = os.getcwd()
# Create object for result export
resultDir = os.path.join(curDir, 'Results')

# Setup variables
cameraRes = (848, 480) # [pixels] from datasheet
chessVertices = (13, 9) # rows, columns
chessSquareSize = 19    # [mm]
imagePoints = []        # 2D
worldPoints = []        # 3D list. Stores the chessboard template for ...
                       # calibration, expanding

# Load images
images = glob.glob('calibrationCaps/*.jpg')

# Termination Criteria for subpixels/corners
iterStop = 30
cornerTolerance = 0.001
criteria = ( (cv2.TERM_CRITERIA_EPS + cv2.TermCriteria_MAX_ITER), ...
             iterStop, cornerTolerance)
winSize = (11, 11)      # OpenCV: Size(5,5), then a (5*2+1)x(5*2+1)=11x11
zeroZone = (-1, -1)      # Deadzone to avoid singularities. (-1,-1) = ...
                       # turned off

# Initialization for World Points (3D) for chessboard.
worldPointsTemplate = numpy.zeros( ( (chessVertices[0]*chessVertices[1]), ...
                                      3), numpy.float32)           # Array (of zeros) with x,y,z coords for ...
                                         # each vertex
worldPointsTemplate[:, :2] = numpy.mgrid[ 0:chessVertices[0], ...
                                         0:chessVertices[1]].T.reshape(-1,2) #Coords for vertices [x_n, y_n, ...
                                         0]. z=0 since flat
worldPointsTemplate *= chessSquareSize ...                                # Convert to [mm]

# ===== Corner Detection and Display =====

for image in images: # Iterates through images

    img = cv2.imread(image)          # Read image from current ...
                                    # index
    grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to greyscale

    ret, corners = cv2.findChessboardCorners(grey, chessVertices, None) # ...
                           # none so Python auto-allocates memory for corners array
```

```

# Want more accurate corner detection - subpixel algorithm
if ret == True:

    # Store corners with whole- and subpixel coordinates
    worldPoints.append(worldPointsTemplate) # Expand list with the ...
                                                template
    cv2.cornerSubPix(grey, corners, winSize, zeroZone, criteria)
    imagePoints.append(corners)           # Expand list with ...
                                                detected corners

    # Display the current image with corner detection
    cv2.drawChessboardCorners(img, chessVertices, corners, ret)
    cv2.imshow('Current Image', img)
    cv2.waitKey(250) # Delay 250 [ms]

cv2.destroyAllWindows()
# ===== Corner Detection and Display =====

# ===== Calibration =====

# Get Camera Matrix and Distortion Coefficients from openCV method
ret, cameraMatrix, distortionCoeffs, rVecs, tVecs = ...
    cv2.calibrateCamera(worldPoints, imagePoints, cameraRes, None, None) # ...
    None: auto-allocates memory

# Display Results
print("\nRMS re-projection error: ", ret) # [pixels]
print("\nCamera Matrix:\n", cameraMatrix)
print("\nDistortion Coefficients:\n", distortionCoeffs)
# print("\nRotation Vectors:\n", rVecs)
# print("\nTranslation Vectors:\n", tVecs)

img = cv2.imread('calibrationCaps/screenshot_0.jpg')
# ^CHANGE IMAGE TO DISPLAY HERE

# ----- Undistortion Display -----
undistortedImg = cv2.undistort(img, cameraMatrix, distortionCoeffs, None)

# Display images
while(True):

    # Display Images
    cv2.imshow('Image: Unprocessed', img)
    cv2.imshow('Image: Undistorted, not cropped', undistortedImg)

    # Stop or save images
    keyPressed = cv2.waitKey(1)
    if keyPressed == ord('s'):
        os.chdir(resultDir)
        screenshot = ...
            cv2.imwrite(filename=f"calibrationResultOriginal.jpg", img=img)
        screenshot = cv2.imwrite(filename=f"calibrationResultCropped.jpg", ...
            img=undistortedImg)
        print('Successfully captured resulting images!')
    elif keyPressed == ord('q'):
        break
# ----- Undistortion Display -----
# ===== Calibration =====

```

### G.1.6 depthCamera.py

```
import cv2                      # OpenCV
import pyrealsense2 as rs        # RealSense wrapper
import numpy                     # Python math
import os                        # For path

# Depth Camera connection
pipe = rs.pipeline()
config = rs.config()
height = 480
width = 848
fps = 60
alpha = 0.15

# Depth reading configuration
cursor = (int(width/2), int(height/2))    # Pixels
curSize = 3                                # Pixels
curThick = 1                                # Pixels
curColor = (255, 255, 255)                  # bgr

# Change directory for screenshot
os.chdir(r'/home/thomaz/MAS306_Drone_G12/Code/Camera/Screenshots')

# Depth Stream
config.enable_stream(rs.stream.depth, width, height, rs.format.z16, fps) # ...
    # (streamType, xRes, yRes, format, fps)
# Infrared Stream
# config.enable_stream(rs.stream.infrared, width, height, rs.format.y8, ...
    # (streamType, xRes, yRes, format, fps)
pipe.start(config)

last10Depth = []
```

```

while(True):

    # Collect frames from camera
    frame = pipe.wait_for_frames()
    frame = frame.get_depth_frame()

    # Convert to numpy array
    image = numpy.asarray(frame.get_data())

    # Read distance at cursor
    depth = image.item(cursor[1], cursor[0])
    print("\nDepth Reading: ", depth)
    print(" [mm]\n")
    last10Depth.append(depth)
    if (len(last10Depth) > 10):
        last10Depth.pop(0)

    # Add color coding
    image = cv2.applyColorMap(cv2.convertScaleAbs(image, alpha=alpha), ...
        cv2.COLORMAP_TURBO)

    # Display cursor
    cv2.circle(image, cursor, curSize, curColor, thickness=curThick)

    # Display the current frame
    cv2.imshow('LiveReading', image)

    # Store key pressed during 1 [ms] delay
    pressedKey = cv2.waitKey(1)

    # S to take screenshot
    if pressedKey == ord('s'):
        cv2.imwrite(filename="screenshotDepth.jpg", img=image) # ...
        Incrementing filename
        print('Screenshot successful!')

    # P to pause
    elif pressedKey == ord('p'):      # Press P to pause
        avgLast10 = sum(last10Depth)/len(last10Depth)
        print(f"\nAverage Past 10 values: {avgLast10}\n")
        cv2.waitKey(-1)

    # Q to stop stream
    elif pressedKey == ord('q'):
        break

pipe.stop()                      # Stop recording
cv2.destroyAllWindows() # Free resources

```

### G.1.7 poseMarkerDepthPreArena.py

```
# ----- Libraries -----
import cv2
import cv2.aruco as aruco # For simplification
import pyrealsense2 as rs
import numpy
# ----- Libraries -----

# ----- Constant variables for simple changes -----
axesLength = 0.01
screenHeight = 480    # pixels
screenWidth = 848     # pixels
fps = 60              # pixels
cColor = (0, 0, 120) # bgr
cThick = 1            # pixels

markerSize = 0.05 # Length of ArUco marker sides [m]

# vvv INTRINSICS ARE FOR 848x480 vvv
    # Distortion Coefficients
distortionCoefficients = numpy.array(
    [ 0.0, 0.0, 0.0, 0.0, 0.0]) # [k1, k2, p1, p2, k3]

    # Intrinsic Camera Matrix
cameraMatrix = numpy.array([
[608.76301751,      0.0,          429.37397121],
[  0.0,           609.23981796, 232.71315263],
[  0.0,           0.0,             1.0        ]])

print("\nCamera Matrix\n", cameraMatrix)
print("\nDistortion Coefficients\n", distortionCoefficients)

# ----- Constant variables for simple changes -----

# Set dictionary for the markers
arucoParams      = aruco.DetectorParameters()
arucoDictionary = aruco.getPredefinedDictionary(aruco.DICT_6X6_50)

# Setup configuration and start pipeline stream
pipe = rs.pipeline()
config = rs.config()

# Color Stream
config.enable_stream(rs.stream.color, screenWidth, screenHeight, ...
    rs.format.bgr8, fps) # (streamType, xRes, yRes, format, fps)
# Depth Stream
config.enable_stream(rs.stream.depth, screenWidth, screenHeight, ...
    rs.format.z16, fps) # (streamType, xRes, yRes, format, fps)
pipe.start(config)
```

```

while(True):

    # Frame Collection
    frame = pipe.wait_for_frames() # collect frames from camera (depth, ...
        color, etc)

    # Color Stream
    color_frame = frame.get_color_frame()      # Extract Color frame
    color_image = numpy.asarray(color_frame.get_data()) # Convert to ...
        NumPy array

    # Depth Stream
    depth_frame = frame.get_depth_frame()      # Extract Depth frame
    depth_image = numpy.asarray(depth_frame.get_data()) # Convert to ...
        NumPy array

    # Marker Identification
    gray = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)    # Grayscale image
    corners, ids, rejectedImagePoints = aruco.detectMarkers(gray, ...
        arucoDictionary, parameters=arucoParams)

    # Is marker detected?
    if len(corners) > 0:

        # Iterate through list of markers
        for (markerCorner, markerID) in zip(corners, ids):

            # Pose reading
            rotVector, transVector, markerPoints = ...
                aruco.estimatePoseSingleMarkers(
                    markerCorner, markerSize, cameraMatrix=cameraMatrix, ...
                    distCoeffs=distortionCoefficients)

            # Draw marker axes
            cv2.drawFrameAxes(color_image, cameraMatrix=cameraMatrix,
                distCoeffs=distortionCoefficients, ...
                    rvec=rotVector, tvec=transVector, ...
                    length=axesLength)

            # Round and display translation vector
            transVector = numpy.around(transVector, 4)
            print("\nTranslation Vector: ", transVector)

            # Extract corners
            corners = markerCorner.reshape(4,2)
            (topLeft, topRight, bottomRight, bottomLeft) = corners

            # Print Current marker ID
            print("Current ID: ", markerID)

            # Extract middle of marker
            avgCorner_x = int((topLeft[0] + topRight[0] + bottomLeft[0] + ...
                bottomRight[0])/4)
            avgCorner_y = int((topLeft[1] + topRight[1] + bottomLeft[1] + ...
                bottomRight[1])/4)

```

```

# Extract depth distance at middle of marker
depthDist = depth_image.item(avgCorner_y, avgCorner_x)

# Display crosshair on Depth Stream
    # Horizontal
cv2.line(depth_image, (0, avgCorner_y), (screenWidth, ...
    avgCorner_y), cColor, cThick)
    # Vertical
cv2.line(depth_image, (avgCorner_x, 0), (avgCorner_x, ...
    screenHeight,), cColor, cThick)

# Print Depth
print("Depth Distance: ", depthDist)

# Depth Stream: Add color map
depth_image = cv2.applyColorMap(cv2.convertScaleAbs(depth_image, ...
    alpha=0.15), cv2.COLORMAP_TURBO)

# Display image
cv2.imshow('ColorStream', color_image)
cv2.imshow('DepthStream', depth_image)

# Loop breaker
pressedKey = cv2.waitKey(1)
if pressedKey == ord('q'):          # Press Q to stop
    break
elif pressedKey == ord('p'):        # Press P to pause
    cv2.waitKey(-1)

pipe.stop()                         # Stop recording
cv2.destroyAllWindows() # Free resources

```

### G.1.8 recordingCamera.py

```
import cv2           # OpenCV
import pyrealsense2 as rs   # RealSense Wrapper
import numpy          # Python Math
import os             # For path

# ----- Recording Setup -----
# Directory to save video
dir = r'/home/thomaz/Recordings'
os.chdir(dir)

# Four-Character Code (File format)
fourcc = cv2.VideoWriter_fourcc(*'MJPG')

# Image Transfer Rate
fps = 60

# Image size [pixels]
w = 848 # Width
h = 480 # Height

# File Configuration for recording
recording = cv2.VideoWriter('recordingPostDepthTest.avi', fourcc, fps, (w,h))
# ----- Recording Setup -----


# Depth Camera connection
pipe = rs.pipeline()
cfg = rs.config()

# Setup Stream
cfg.enable_stream(rs.stream.color, w, h, rs.format.bgr8, fps) # ...
# (streamType, xRes, yRes, format, fps)
pipe.start(cfg)

while(True):
    # Extract and convert frame
    frame = pipe.wait_for_frames() # waits for and collects all frames ...
    # from camera (depth, color, etc)
    color_frame = frame.get_color_frame()
    color_image = numpy.asarray(color_frame.get_data())

    # Save frame to file
    recording.write(color_image)

    # In case of too large size to display
    #color_image = cv2.resize(color_image, displaySize)

    # Display the current frame
    cv2.imshow('LiveReading', color_image)

    # Press Q to stop recording
    if cv2.waitKey(1) == ord('q'):
        break

recording.release()
pipe.stop()          # Stop recording
cv2.destroyAllWindows() # Free resources
```

### G.1.9 markerTestAnalyzer.py

```
import pyrealsense2 as rs      # stream configuration
import cv2                     # Show video with OpenCV
import cv2.aruco as aruco     # Simplification
import os                      # Check file-extension
import numpy                   # Python Math
import time                    # Computation Timer

axesLength = 0.01

# Length of ArUco marker sides
markerSize = 0.05 # [m]

# Imported from Camera Calibration
distortionCoefficients = numpy.array(
    [ 0.0, 0.0, 0.0, 0.0, 0.0]) # [k1, k2, p1, p2, k3]
cameraMatrix = numpy.array([
[608.76301751,      0.0,        429.37397121],
[  0.0,       609.23981796, 232.71315263],
[  0.0,           0.0,         1.0        ]])

# Directory to read/write videos
dir = r'/home/thomaz/Recordings'
os.chdir(dir)

# Create object for parameters
arucoParams = aruco.DetectorParameters()

# Set dictionary for the markers
dictList = numpy.array([
    aruco.DICT_4X4_50,
    aruco.DICT_5X5_50,
    aruco.DICT_6X6_50,
    aruco.DICT_7X7_50,
    aruco.DICT_ARUCO_ORIGINAL,
    aruco.DICT_APRILTAG_16h5,
    aruco.DICT_APRILTAG_25h9,
    aruco.DICT_APRILTAG_36h10,
    aruco.DICT_APRILTAG_36h11,
    aruco.DICT_ARUCO_MIP_36h12
])

# Size of window to display recording
displaySize = (960, 540)

# Store relevant values
framesWithData = []      # Array to store frames with data
computationTime = []     # Array for computation times
```

```

for dict in dictList:
    # Restart variables for relevant values
    startTimer = time.time() # Start timer per dict
    curDataFrames = 0          # Frames with data for current dict
    totalFrames = 0           # Total Frames extraction part 1

    # Fetch current dictionary
    dictionary = aruco.getPredefinedDictionary(dict)

    # Import recording for each dictionary
    recording = cv2.VideoCapture('recordingPostDepthTest.avi')

    while(recording.isOpened()):

        # Extract frames
        ret, frame = recording.read()

        # Total Frames extraction part 2
        totalFrames += 1

        # Stop while loop if no more frames
        if not ret:
            break

        # Identification
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)      # Grayscale image
        corners, ids, rejectedImagePoints = aruco.detectMarkers(gray, ...
            dictionary, parameters=arucoParams)

        if len(corners) > 0:
            for i in range(0, len(ids)):

                # Pose Estimate using ArUco
                rotVector, transVector, markerPoints = ...
                    aruco.estimatePoseSingleMarkers(
                        corners[i], markerSize, cameraMatrix=cameraMatrix, ...
                        distCoeffs=distortionCoefficients)

                # Number of frames with data
                curDataFrames += 1

            # Press Q to stop video playback
            if cv2.waitKey(1) == ord('q'):
                break

        # Release playback after each dictionary
        recording.release()

    # Computation Time
    endTimer = time.time()                      # Time of loop stop
    currentTime = endTimer - startTimer # Time difference
    computationTime.append(currentTime) # Increment array

    # Number of Frames with Data
    framesWithData.append(curDataFrames)

# Display relevant array values
print("\nComputation Time:\n", computationTime)
print("\nFrames With Data:\n", framesWithData)
print("\nTotal Frames: ", totalFrames)

```

### G.1.10 markerTestVideoExport.py

```
import pyrealsense2 as rs      # stream configuration
import cv2                      # Show video with OpenCV
import cv2.aruco as aruco       # Simplification
import os                        # Check file-extension
import numpy                     # Python Math

font = cv2.FONT_HERSHEY_SIMPLEX
fontColor = (200, 20, 200)
fontScale = 0.5
fontThickness = 1
axesLength = 0.01

# Length of ArUco marker sides
markerSize = 0.04 # [m]

# Imported from Camera Calibration, hardcoded for testing first
cameraMatrix = numpy.array([
    [1379.333496, 0,         973.144470],      # [f_x, 0.0, c_x] used ...
    principal points
    [ 0,          1378.661255, 535.200500],      # [0.0, f_y, c_y] as ...
    optical center points
    [ 0,          0,           1.0 ] ])   # [0.0, 0.0, 1.0]
distortionCoefficients = numpy.array(
    [ 0.0, 0.0, 0.0, 0.0, 0.0]) # [k1, k2, p1, p2, k3]

# ----- Import/Export Video Setup -----
# Directory to save video
dir = r'/home/thomaz/Recordings'
os.chdir(dir)

# Four-Character Code (File format)
fourcc = cv2.VideoWriter_fourcc(*'MJPG')

# Image Transfer Rate
fps = 60

# Image size [pixels]
w = 848 # Width
h = 480 # Height
# ----- Import/Export Video Setup -----

# Create object for parameters
arucoParams = aruco.DetectorParameters()

# Set dictionary for the markers
dictList = numpy.array([
    aruco.DICT_4X4_50,
    aruco.DICT_5X5_50,
    aruco.DICT_6X6_50,
    aruco.DICT_7X7_50,
    aruco.DICT_ARUCO_ORIGINAL,
    aruco.DICT_APRILTAG_16h5,
    aruco.DICT_APRILTAG_25h9,
    aruco.DICT_APRILTAG_36h10,
    aruco.DICT_APRILTAG_36h11,
    aruco.DICT_APRILTAG_36h12
])
```

```

# Video number for filename
vidNr = 0

# Iterate through dict list
for dict in dictList:

    # Start new recording
    vidNr += 1
    resultVid = cv2.VideoWriter(f'resultsTestPostDepth_{vidNr}.avi', ...
        fourcc, fps, (w,h))

    # Fetch current dictionary
    dictionary = aruco.getPredefinedDictionary(dict)

    # Import recording
    recording = cv2.VideoCapture('recordingPostDepthTest.avi')

    while(recording.isOpened()):

        # Extract frames
        ret, frame = recording.read()

        # Stop while loop if no more frames
        if not ret:
            break

        # Identification
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)      # Grayscale image
        corners, ids, rejectedImagePoints = aruco.detectMarkers(gray, ...
            dictionary, parameters=arucoParams)

        if len(corners) > 0:
            for i in range(0, len(ids)):

                # Pose Estimate using ArUco
                rotVector, transVector, markerPoints = ...
                    aruco.estimatePoseSingleMarkers(
                        corners[i], markerSize, cameraMatrix=cameraMatrix, ...
                        distCoeffs=distortionCoefficients)

                # Draw axes
                cv2.drawFrameAxes(frame, cameraMatrix=cameraMatrix,
                    distCoeffs=distortionCoefficients, ...
                    rvec=rotVector, tvec=transVector, ...
                    length=axesLength)

        # Display the current frame
        cv2.imshow('Playback', frame)  # Display the current frame

        # Export results video
        resultVid.write(frame)  # Incrementing filename

        # Press Q to stop video playback
        if cv2.waitKey(1) == ord('q'):
            break

    # Release playback after each dictionary
    recording.release()
    resultVid.release()

```

### G.1.11 computerVisionMain.py

This code uses the Module G.5.1 in a folder named comm\_module

```
# ----- Libraries -----
import cv2                                     # OpenCV
import cv2.aruco as aruco                      # For simplification
import pyrealsense2 as rs                        # RealSense Wrapper
import numpy                                     #
import time
from comm_module import dronePosVec_pb2 #should be ...
    protobuf.dronePosVec_pb2 but wont work for some reason
from comm_module import arenaComm

# ----- Libraries -----
# ----- Socket Class -----


#START MULTIPROCESS
hostname = 'b12.uia.no'
messager = ...
    arenaComm.ArenaCommunication(arenaComm.SendrecvType.GENERICSEND, ...
    hostname, (3,1), (3,3),dronePosVec_pb2.camera)

# ----- Socket Class -----
# ---- Constant variables for simple changes -----
axesLength = 0.1
screenHeight = 480    # pixels
screenWidth = 848     # pixels
fps = 60              # pixels
#cColor = (0, 0, 120) # bgr
#cThick = 1           # pixels

# Physical marker sizes
markerSize = 0.05 # Marker sides [m]
markerPoints = numpy.array([[-markerSize / 2, markerSize / 2, 0],
                           [markerSize / 2, markerSize / 2, 0],
                           [markerSize / 2, -markerSize / 2, 0],
                           [-markerSize / 2, -markerSize / 2, 0]], ...
                           dtype=numpy.float32)

#defaultRightCorner = numpy.array([(markerSize/2)*(1+1-1)/3)*1000,
##                                ((markerSize/2)*(1+1-1)/3)*1000, 0.0])

# vvv INTRINSICS ARE FOR 848x480 vvv
    # Distortion Coefficients
distortionCoefficients = numpy.array(
    [ 0.0, 0.0, 0.0, 0.0, 0.0]) # [k1, k2, p1, p2, k3]

    # Intrinsic Camera Matrix
fx = 608.76301751
fy = 609.23981796
cx = 429.37397121
cy = 232.71315263
cameraMatrix = numpy.array([
[ fx, 0.0, cx ],
[ 0.0, fy, cy ],
[ 0.0, 0.0, 1.0]])

print("\nCamera Matrix\n", cameraMatrix)
print("\nDistortion Coefficients\n", distortionCoefficients)
```

```

# ----- Constant variables for simple changes -------

# Set dictionary for the markers
arucoParams      = aruco.DetectorParameters()
arucoDictionary = aruco.getPredefinedDictionary(aruco.DICT_5X5_50)

# Setup configuration and start pipeline stream
pipe = rs.pipeline()
config = rs.config()

# Color Stream
config.enable_stream(rs.stream.color, screenWidth, screenHeight, ...
    rs.format.bgr8, fps) # (streamType, xRes, yRes, format, fps)
# Depth Stream
#config.enable_stream(rs.stream.depth, screenWidth, screenHeight, ...
#    rs.format.z16, fps) # (streamType, xRes, yRes, format, fps)
#config.enable_stream(rs.stream.infrared, screenWidth, screenHeight, ...
#    rs.format.y8, fps) # (streamType, xRes, yRes, format, fps)
pipe.start(config)

rotVectors = []
transVectors = []
reprojError = 0
transVectorsExp = numpy.empty(messenger.posShape[0] * ...
    messenger.posShape[1], dtype=float) #exp = export, needed because tuple

while(True):

    # Frame Collection
    frame = pipe.wait_for_frames()                      # collect frames from camera ...
        (depth, color, etc)
    messenger.dp.timestamp_ns = time.time_ns() #timestamp of when the image ...
        is taken
    # Color Stream
    color_frame = frame.get_color_frame()                 # Extract Color ...
        frame
    color_image = numpy.asarray(color_frame.get_data()) # Convert to ...
        NumPy array

    # Depth Stream
    #depth_frame = frame.get_depth_frame()      # Extract Depth frame
    #depth_image = numpy.asarray(depth_frame.get_data()) # Convert to ...
        NumPy array

    # Marker Identification
    gray = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)    # Grayscale image
    corners, ids, rejectedImagePoints = aruco.detectMarkers(gray, ...
        arucoDictionary, parameters=arucoParams)

    ##### Export Depth and Color frames here #####
    #print("\nDepth Image: ", depth_image)

    # Is marker detected?
    if len(corners) > 0:

        # Iterate through list of markers
        for (markerCorner, markerID) in zip(corners, ids):

```

```

# SolvePnPGeneric for extracting all possible vectors
retVal, rotVectors, transVectors, reprojError = ...
    cv2.solvePnPGeneric(
        markerPoints, markerCorner, cameraMatrix, ...
            distortionCoefficients, rvecs=rotVectors, ...
                tvecs=transVectors, reprojectionError=reprojError,
                    useExtrinsicGuess=False, flags=cv2.SOLVEPNP_IPPE)

# Print current vectors
#print("\nRotation Vectors: ", rotVectors)
#print("\nTranslation Vectors: ", transVectors)

# Extract Rotation Matrices
rMat1, _ = cv2.Rodrigues(rotVectors[0])
rMat2, _ = cv2.Rodrigues(rotVectors[1])
#print("Rotation Matrix: ", rMat)

# Reprojection Error Logging
#print("\nReprojection Error: ", reprojError)
#write.writerow(reprojError)

# Draw marker axes
cv2.drawFrameAxes(color_image, cameraMatrix=cameraMatrix,
                   distCoeffs=distortionCoefficients, ...
                       rvec=rotVectors[0], tvec=transVectors[0], ...
                           length=axesLength)

#flatten rotation:
for i in range(messager.posShape[0]):
    transVectorsExp[i] = transVectors[0][i][0]

messager.dp.rotation[:] = rMat1.flatten()
messager.dp.rotation2[:] = rMat2.flatten()
messager.dp.position[:] = transVectorsExp

# Depth Stream: Add color map
#depth_image = cv2.applyColorMap(cv2.convertScaleAbs(depth_image, ...
#    alpha=0.15), cv2.COLORMAP_TURBO)

# Display image
#cv2.imshow('DepthStream', depth_image)
cv2.imshow('ColorStream', color_image)

# Loop breaker
pressedKey = cv2.waitKey(1)
if pressedKey == ord('q'):          # Press Q to stop
    break
elif pressedKey == ord('p'):        # Press P to pause
    cv2.waitKey(-1)

messager.addPostToSendQueue()
del messager.dp.rotation[:]
del messager.dp.rotation2[:]
del messager.dp.position[:]

pipe.stop()                      # Stop recording
cv2.destroyAllWindows() # Free resources
messager.endmps()

```

## G.2 Pose Validation Test

### G.2.1 poseTestInitializer.py

```
import cv2
import cv2.aruco as aruco # For simplification
import pyrealsense2 as rs
import numpy

# ----- Constant variables for simple changes -----
axesLength = 0.1
screenHeight = 480    # pixels
screenWidth = 848     # pixels
fps = 60              # pixels
tolerance = 5         # degrees

# Rotation Matrix
rotMatCalib = numpy.array([
    [-1.0, 0.0, 0.0],
    [ 0.0, 1.0, 0.0],
    [ 0.0, 0.0,-1.0]
])
# Physical marker sizes
markerSize = 0.167 # Marker sides [m]
markerPoints = numpy.array([[-markerSize / 2, markerSize / 2, 0],
                           [markerSize / 2, markerSize / 2, 0],
                           [markerSize / 2, -markerSize / 2, 0],
                           [-markerSize / 2, -markerSize / 2, 0]], ...
                           dtype=numpy.float32)
# Distortion Coefficients
distortionCoefficients = numpy.array(
    [ 0.0, 0.0, 0.0, 0.0, 0.0]) # [k1, k2, p1, p2, k3]

# Intrinsic Camera Matrix
fx = 608.76301751
fy = 609.23981796
cx = 429.37397121
cy = 232.71315263
cameraMatrix = numpy.array([
[ fx, 0.0, cx ],
[ 0.0, fy, cy ],
[ 0.0, 0.0, 1.0]])
# ----- Constant variables for simple changes -----

# Set dictionary for the markers
arucoParams      = aruco.DetectorParameters()
arucoDictionary = aruco.getPredefinedDictionary(aruco.DICT_5X5_50)

# Setup configuration and start pipeline stream
pipe = rs.pipeline()
config = rs.config()

# Color Stream
config.enable_stream(rs.stream.color, screenWidth, screenHeight, ...
    rs.format.bgr8, fps) # (streamType, xRes, yRes, format, fps)
pipe.start(config)
rotVectors = []
transVectors = []
reprojError = 0
```

```

while(True):
    # Frame Collection
    frame = pipe.wait_for_frames()           # collect frames from camera ...
    (depth, color, etc)

    # Color Stream
    color_frame = frame.get_color_frame()      # Extract Color frame
    color_image = numpy.asarray(color_frame.get_data()) # Convert to ...
    NumPy array

    # Marker Identification
    gray = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)    # Grayscale image
    corners, ids, rejectedImagePoints = aruco.detectMarkers(gray, ...
        arucoDictionary, parameters=arucoParams)

    # Is marker detected?
    if len(corners) > 0:

        # Iterate through list of markers
        for (markerCorner, markerID) in zip(corners, ids):

            # SolvePnPGeneric for extracting all possible vectors
            retVal, rotVectors, transVectors, reprojError = ...
                cv2.solvePnPGeneric(
                    markerPoints, markerCorner, cameraMatrix, ...
                        distortionCoefficients, rvecs=rotVectors, ...
                            tvecs=transVectors, reprojectionError=reprojError,
                            useExtrinsicGuess=False, flags=cv2.SOLVEPNP_IPPE)

            # Draw marker axes
            cv2.drawFrameAxes(color_image, cameraMatrix=cameraMatrix,
                distCoeffs=distortionCoefficients, ...
                    rvec=rotVectors[0], tvec=transVectors[0], ...
                        length=axesLength)

            # Calculate angle difference
            curRotMat, _ = cv2.Rodrigues(rotVectors[0])
            print("\nCurrent Rotmat: ", curRotMat)
            rotMatDiff = numpy.dot(curRotMat, rotMatCalib.T)
            rotVecDiff, _ = cv2.Rodrigues(rotMatDiff)
            diffAngle = numpy.rad2deg(numpy.linalg.norm(rotVecDiff))

            print("\nDiffAngle: ", diffAngle)

            if (diffAngle < tolerance):
                print("\nStop rotating, inside tolerance.")

        # Display image
        cv2.imshow('ColorStream', color_image)

        # Loop breaker
        pressedKey = cv2.waitKey(1)
        if pressedKey == ord('q'):          # Press Q to stop
            break
        elif pressedKey == ord('p'):        # Press P to pause
            cv2.waitKey(-1)

    pipe.stop()                      # Stop recording
    cv2.destroyAllWindows() # Free resources

```

## G.2.2 cameraCenter.py

```
# ----- Libraries -----
import cv2
import pyrealsense2 as rs
import numpy
# ----- Libraries -----


screenHeight = 480      # pixels
screenWidth = 848       # pixels
fps = 60                 # frames/second
cColor = (0, 0, 120)    # bgr
cThick = 1               # pixels


# Setup configuration and start pipeline stream
pipe = rs.pipeline()
config = rs.config()

config.enable_stream(rs.stream.color, screenWidth, screenHeight, ...
    rs.format.bgr8, fps) # (streamType, xRes, yRes, format, fps)
pipe.start(config)

while(True):


    # Frame Collection
    frame = pipe.wait_for_frames()           # collect frames from camera ...
    (depth, color, etc)
    color_frame = frame.get_color_frame()    # Extract RGB module frame
    color_image = numpy.asarray(color_frame.get_data()) # Convert to ...
    NumPy array

    # Display crosshair horizontal then vertical
    cv2.line(color_image, (0, int(screenHeight/2)), (screenWidth, ...
        int(screenHeight/2)), cColor, cThick)
    cv2.line(color_image, (int(screenWidth/2), 0), (int(screenWidth/2), ...
        screenHeight), cColor, cThick)

    # Display image
    cv2.imshow('LiveReading', color_image)

    # Loop breaker
    pressedKey = cv2.waitKey(1)
    if pressedKey == ord('q'):            # Press Q to stop
        break
    elif pressedKey == ord('p'):          # Press P to pause
        cv2.waitKey(-1)

pipe.stop()                  # Stop recording
cv2.destroyAllWindows() # Free resources
```

### G.2.3 poseTestRecording.py

```
import cv2           # OpenCV
import pyrealsense2 as rs   # RealSense Wrapper
import numpy          # Python Math
import os             # For path
import csv
import time

# ----- Recording Setup -----
# Directory to save video
dir = r'/home/thomaz/poseValidationTest'
os.chdir(dir)

# Four-Character Code (File format)
fourcc = cv2.VideoWriter_fourcc(*'MJPG')

# Image Transfer Rate
fps = 60

# Image size [pixels]
w = 848 # Width
h = 480 # Height

# File Configuration for recording
items = os.listdir(dir)
recNum = round(len(items)/2, None)
recording = cv2.VideoWriter(f'qualisysTest_{recNum}.avi', fourcc, fps, (w,h))
# ----- Recording Setup -----


# Depth Camera connection
pipe = rs.pipeline()
cfg = rs.config()

# Setup csv file
filename = f"cameraTimestamps_{recNum}.csv"
fields = ['Frame', 'Timestamp']

# Column names
with open(filename, 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)

# Setup Stream
cfg.enable_stream(rs.stream.color, w, h, rs.format.bgr8, fps) # ...
    (streamType, xRes, yRes, format, fps)
pipe.start(cfg)
n = 1
```

```

with open(filename, 'a') as csvfile:
    # Set writer for loop
    csvwriter = csv.writer(csvfile)
    startTime = time.monotonic()
    while(True):
        # Extract and convert frame
        frame = pipe.wait_for_frames() # waits for and collects all frames ...
        from camera (depth, color, etc)
        csvwriter.writerow([n,time.monotonic()-startTime])
        n += 1

        color_frame = frame.get_color_frame()
        color_image = numpy.asarray(color_frame.get_data())

        # Save frame to file
        if True:
            recording.write(color_image)

        # Display the current frame
        cv2.imshow('LiveReading', color_image)

        # Press Q to stop recording
        if cv2.waitKey(1) == ord('q'):
            break

recording.release()
pipe.stop()           # Stop recording
cv2.destroyAllWindows() # Free resources

```

#### G.2.4 poseTestArucoExport.py

```
# ----- Libraries -----
import cv2 # Show video with OpenCV
import cv2.aruco as aruco # Simplification
import os # Check file-extension
import numpy # Python Math
import csv # Data import
# ----- Libraries -----

# ---- Constant variables for simple changes ----
axesLength = 0.1
screenHeight = 480 # pixels
screenWidth = 848 # pixels
fps = 60 # pixels

# Physical marker sizes
markerSize = 0.05 # Marker sides [m]
markerPoints = numpy.array([[-markerSize / 2, markerSize / 2, 0],
                           [markerSize / 2, markerSize / 2, 0],
                           [markerSize / 2, -markerSize / 2, 0],
                           [-markerSize / 2, -markerSize / 2, 0]], ...
                           dtype=numpy.float32)

# vvv INTRINSICS ARE FOR 848x480 vvv
# Distortion Coefficients
distortionCoefficients = numpy.array(
    [ 0.0, 0.0, 0.0, 0.0, 0.0]) # [k1, k2, p1, p2, k3]

# Intrinsic Camera Matrix
fx = 608.76301751
fy = 609.23981796
cx = 429.37397121
cy = 232.71315263
cameraMatrix = numpy.array([
    [ fx, 0.0, cx ],
    [ 0.0, fy, cy ],
    [ 0.0, 0.0, 1.0]])

# ---- Constant variables for simple changes ----

# ----- Import/Export Setup -----
# Directory to save video
dir = r'/home/thomaz/poseValidationTest'
os.chdir(dir)

# Four-Character Code (File format)
fourcc = cv2.VideoWriter_fourcc(*'MJPG')

# Image Transfer Rate
fps = 60

# Image size [pixels]
w = 848 # Width
h = 480 # Height

# Test number for easy change
testNr = 4
# ----- Import/Export Setup -----
```

```

# Create object for parameters
arucoParams = aruco.DetectorParameters()

# Set dictionary for the markers
arucoParams      = aruco.DetectorParameters()
arucoDictionary = aruco.getPredefinedDictionary(aruco.DICT_5X5_50)

rotVectors = []
transVectors = []
reprojError = 0
loopRound = 0

# Start new recording
resultVid = cv2.VideoWriter(f'resultsPoseTest_{testNr}.avi', fourcc, fps, ...
    (w,h))

# Import recording
recording = cv2.VideoCapture(f'qualisysTest_{testNr}.avi')

# Import timestamps from D435 recording
fileOpenCV = open(f'cameraTimestamps_{testNr}.csv')
dataOpenCVreader = csv.reader(fileOpenCV)
dataOpenCV = []
for row in dataOpenCVreader:
    dataOpenCV.append(row)

timestampOpenCV = [row[1] for row in dataOpenCV[1:]]
timestampOpenCV = numpy.array(timestampOpenCV, dtype=float)
iterOpenCV = [row[0] for row in dataOpenCV[1:]]
iterOpenCV = numpy.array(iterOpenCV, dtype=int)

#print("\n dataOpenCV: ", dataOpenCV)
print("\n TimestampsOpenCV: ", timestampOpenCV)

# Setup csv file
filename = f"ExportedAruco_{testNr}.csv"
fields = ['Frame', 'TimeCV', 'tVec0_0', 'tVec0_1', 'tVec0_2', 'tVec1_0', ...
    'tVec1_1', 'tVec1_2',
    'rVec0_0', 'rVec0_1', 'rVec0_2', 'rVec1_0', 'rVec1_1', ...
    'rVec1_2', 'reprojError0', 'reprojError1']

# Column names
with open(filename, 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)

with open(filename, 'a') as csvfile:
    # Set writer for loop
    csvwriter = csv.writer(csvfile)

    while(recording.isOpened()):

        # Extract frames
        ret, frame = recording.read()

        # Stop while loop if no more frames
        if not ret:
            break

```

```

# Identification
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)      # Grayscale image
corners, ids, rejectedImagePoints = aruco.detectMarkers(gray, ...
    arucoDictionary, parameters=arucoParams)

# Is marker detected?
if len(corners) > 0:

    # Iterate through list of markers
    for (markerCorner, markerID) in zip(corners, ids):

        # SolvePnPGeneric for extracting all possible vectors
        retVal, rotVectors, transVectors, reprojError = ...
            cv2.solvePnPGeneric(
                markerPoints, markerCorner, cameraMatrix, ...
                distortionCoefficients, rvecs=rotVectors, ...
                tvecs=transVectors, reprojectionError=reprojError,
                useExtrinsicGuess=False, flags=cv2.SOLVEPNP_IPPE)

        # Draw marker axes
        cv2.drawFrameAxes(frame, cameraMatrix=cameraMatrix,
            distCoeffs=distortionCoefficients, ...
            rvec=rotVectors[0], ...
            tvec=transVectors[0], length=axesLength)

    # Export results csv
    csvwriter.writerow([loopRound, timestampOpenCV[loopRound],
        transVectors[0][0][0], ...
        transVectors[0][1][0], ...
        transVectors[0][2][0],
        transVectors[1][0][0], ...
        transVectors[1][1][0], ...
        transVectors[1][2][0],
        rotVectors[0][0][0], ...
        rotVectors[0][1][0], ...
        rotVectors[0][2][0],
        rotVectors[1][0][0], ...
        rotVectors[1][1][0], ...
        rotVectors[1][2][0], ...
        reprojError[0][0], reprojError[1][0]]))

    # Display the current frame
    cv2.imshow('Playback', frame) # Display the current frame

    # Export results video
    resultVid.write(frame) # Incrementing filename

    # Press Q to stop video playback
    if cv2.waitKey(1) == ord('q'):
        break

    loopRound += 1

# Release playback after each dictionary
recording.release()
resultVid.release()

```

## G.2.5 poseTestSynchExport.py

```
# ----- Libraries -----
import cv2           # Show video with OpenCV
import cv2.aruco as aruco   # Simplification
import os            # Check file-extension
import numpy          # Python Math
import csv            # Data import
# ----- Libraries -----

# Directory to save video
dir = r'/home/thomaz/poseValidationTest'
os.chdir(dir)

# Test number for easy change
testNr = 4

tolerance = 0.1 # meters

rotVectors = []
transVectors = []
reprojError = []
startTime = 0
last10mag = []
detectedStart = False

# Import recording
recording = cv2.VideoCapture(f'qualisysTest_{testNr}.avi')

# Import timestamps from D435 recording
fileOpenCV = open(f'ExportedAruco_{testNr}.csv')
dataOpenCVreader = csv.reader(fileOpenCV)
dataOpenCV = []
for row in dataOpenCVreader:
    dataOpenCV.append(row)

timestampOpenCV = [row[1] for row in dataOpenCV[1:]]
timestampOpenCV = numpy.array(timestampOpenCV, dtype=float)
iterOpenCV = [row[0] for row in dataOpenCV[1:]]
iterOpenCV = numpy.array(iterOpenCV, dtype=int)

# ----- Qualisys Data -----
### Import data from Qualisys ####
fileQTM = open(f'qualisysTest{testNr}_6D.tsv')
QTMreader = csv.reader(fileQTM, delimiter="\t")
QTMdata = []
for row in QTMreader:
    QTMdata.append(row)

#### Translation #####
# Extract all rows of columns 3 to 5 (excluding headers)
transQTM = [row[5:8] for row in QTMdata[14:]]
transQTM = numpy.array(transQTM, dtype=float)/1000.0 # millimeters -> meters

# Measured physical distances from RGB camera to Qualisys L-Frame
pTransVec = numpy.array([0.190, 0.143, 1.564]) # meters
pTransArr = numpy.tile(pTransVec, (len(transQTM), 1))
```

```

# Correct translation
transQTM = pTransArr - transQTM

##### Start Detection #####
detectedStartQTM = False
last10magQTM = []

# Time list
timeQTM = [row[1] for row in QTMdata[14:]]
timeQTM = numpy.array(timeQTM, dtype=float)
iterQTM = [row[0] for row in QTMdata[14:]]
iterQTM = numpy.array(iterQTM, dtype=int)

##### Find start time QTM #####
for i, row in enumerate(transQTM):

    # Last 10 magnitudes
    curTransMagQTM = numpy.linalg.norm(transQTM[i])
    last10magQTM.append(curTransMagQTM)
    if (len(last10magQTM) > 10):
        last10magQTM.pop(0)

    # Movement detection
    if (not detectedStartQTM) and (len(last10magQTM) == 10) and ...
       (abs(curTransMagQTM - last10magQTM[0]) > tolerance):
        startTimeQTM = timeQTM[i]
        detectedStartQTM = True

# Start value to be changed
correctTimeQTM = startTimeQTM

# Difference list -----> starts @ 1 not 0 <-----
dTimeQTM = [timeQTM[i]-timeQTM[i-1] for i in range(1, len(timeQTM))]

##### Rotation #####
# Extract all rows of columns 10 to 18 (excluding headers)
rotElementsQTM = [row[12:21] for row in QTMdata[14:]]
rotElementsQTM = [[float(element) for element in sublist] for sublist in ...
                  rotElementsQTM]

# Rotation Matrix for relating frames: Qualisys -> D435
rotMatQTM2OpenCV = numpy.array([
    [-1.0, 0.0, 0.0],
    [0.0, 1.0, 0.0],
    [0.0, 0.0, -1.0]
])

# Convert to new form and relate to D435 frame
rotMatQTM = []
for row in rotElementsQTM:
    matrix = numpy.array(row).reshape(3,3)
    matrix = matrix @ rotMatQTM2OpenCV # Convert orientation
    rotMatQTM.append(matrix)
# ----- Qualisys Data -----

```

```

# Setup csv file
filename = f"ExportedResults_{testNr}.csv"
fields = ['Time', 'xQTM', 'yQTM', 'zQTM', 'xCV0', 'yCV0', 'zCV0',
          'v0R11', 'v0R12', 'v0R13', 'v0R21', 'v0R22', 'v0R23', 'v0R31', ...
          'v0R32', 'v0R33',
          'v1R11', 'v1R12', 'v1R13', 'v1R21', 'v1R22', 'v1R23', 'v1R31', ...
          'v1R32', 'v1R33',
          'qtmR11', 'qtmR12', 'qtmR13', 'qtmR21', 'qtmR22', 'qtmR23', ...
          'qtmR31', 'qtmR32', 'qtmR33', 'xCV1', 'yCV1', 'zCV1',]

# Column names
with open(filename, 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)

with open(filename, 'a') as csvfile:
    # Set writer for loop
    csvwriter = csv.writer(csvfile)

    # From 0 to end(iterOpenCV)
    for loopRound in range(len(iterOpenCV)):
        ##### THIS IS DONE INSIDE DETECTION #####
        # Create transVectors
        transVector0 = numpy.array(dataOpenCV[loopRound+1][2:5], ...
                                   dtype=numpy.float32)
        transVector1 = numpy.array(dataOpenCV[loopRound+1][5:8], ...
                                   dtype=numpy.float32)
        # Create rotVectors
        rotVector0 = numpy.array(dataOpenCV[loopRound+1][8:11], ...
                                   dtype=numpy.float32)
        rotVector1 = numpy.array(dataOpenCV[loopRound+1][11:14], ...
                                   dtype=numpy.float32)

        ### Find start time OpenCV ###
        # Save current magnitude
        curTransMag = numpy.linalg.norm(transVector0)

        # Movement detection
        if (not detectedStart) and (len(last10mag) == 10) and ...
           (abs(last10mag[0] - curTransMag) > tolerance):
            startTime = timestampOpenCV[loopRound]
            startLoop = loopRound
            detectedStart = True

        # Save last 10 frames magnitude
        last10mag.append(curTransMag)
        if (len(last10mag) > 10):
            last10mag.pop(0)

        # Current timestamp
        curTime = timestampOpenCV[loopRound] - startTime

        # Match closest timestamps
        if detectedStart:
            # QTM time match - i will be saved when break
            for i in range(len(timeQTM)):
                if (0 < ((timeQTM[i] - startTimeQTM) - curTime)):
                    correctTimeQTM = timeQTM[i]
                    break

```

```

# Convert to rotation matrices
rotMat0, _ = cv2.Rodrigues(rotVector0)
rotMat1, _ = cv2.Rodrigues(rotVector1)

# Export time and translation vectors
csvwriter.writerow([curTime, transQTM[i][0], transQTM[i][1], ...
    transQTM[i][2],
    transVector0[0], transVector0[1], ...
    transVector0[2],
    rotMat0[0][0], rotMat0[0][1], rotMat0[0][2],
    rotMat0[1][0], rotMat0[1][1], rotMat0[1][2],
    rotMat0[2][0], rotMat0[2][1], rotMat0[2][2],
    rotMat1[0][0], rotMat1[0][1], rotMat1[0][2],
    rotMat1[0][1], rotMat1[1][1], rotMat1[1][2],
    rotMat1[0][2], rotMat1[2][1], rotMat1[2][2],
    rotMatQTM[i][0][0], rotMatQTM[i][0][1], ...
    rotMatQTM[i][0][2],
    rotMatQTM[i][1][0], rotMatQTM[i][1][1], ...
    rotMatQTM[i][1][2],
    rotMatQTM[i][2][0], rotMatQTM[i][2][1], ...
    rotMatQTM[i][2][2],
    transVector1[0], transVector1[1], ...
    transVector1[2],])

# transVector1 was added later, hence it's last placement to ...
# avoid plot script redesign

##### THIS IS DONE INSIDE DETECTION #####
print("\nStartOpen: ", startTime)
print("\nStartQTM: ", startTimeQTM)
print("\nStartLoop: ", startLoop)

```

### G.2.6 qualisysPlotting.m

```
clc; clear; close all;

%% Import Results from Pose Validation Test

testNr = 4;
fileName = ['ExportedResults_', num2str(testNr), '.csv'];
data = csvread(fileName, 1,0);

time = data(:,1);
xQTM = data(:,2);
yQTM = data(:,3);
zQTM = data(:,4);
xCV0 = data(:,5);
yCV0 = -data(:,6); % Flip axis
zCV0 = data(:,7);
xCV1 = data(:,35);
yCV1 = -data(:,36); % Flip axis
zCV1 = data(:,37);

trans = [0.190 0.143 1.564]; % Physically measured in m

% X Limits for Translation Plots [seconds]
startTimeTrans = 0;
stopTimeTrans = 47;

% X Limits for Rotation Plots [seconds]
startTimeRot = 0;
stopTimeRot = 47;

%% Translation Analysis

% Initialize difference lists
diffxV01 = zeros(length(time), 1);
diffyV01 = zeros(length(time), 1);
diffzV01 = zeros(length(time), 1);
diffxCV0QTM = zeros(length(time), 1);
diffyCV0QTM = zeros(length(time), 1);
diffzCV0QTM = zeros(length(time), 1);
diffxCV1QTM = zeros(length(time), 1);
diffyCV1QTM = zeros(length(time), 1);
diffzCV1QTM = zeros(length(time), 1);

% Append differences to lists
for i = 1 : length(time)
    if (xQTM(i) ~= trans(1))
        diffxV01(i) = abs(xCV0(i) - xCV1(i));
        diffxCV0QTM(i) = abs(xCV0(i) - xQTM(i));
        diffxCV1QTM(i) = abs(xCV1(i) - xQTM(i));
    end
    if (yQTM(i) ~= trans(2))
        diffyV01(i) = abs(yCV0(i) - yCV1(i));
        diffyCV0QTM(i) = abs(yCV0(i) - yQTM(i));
        diffyCV1QTM(i) = abs(yCV1(i) - yQTM(i));
    end
end
```

```

if (zQTM(i) ~= trans(3))
    diffzV01(i)      = abs(zCV0(i) - zCV1(i));
    diffzCV0QTM(i)   = abs(zCV0(i) - zQTM(i));
    diffzCV1QTM(i)   = abs(zCV1(i) - zQTM(i));
end
format short
% Calculate average difference from list - mm
avgDiffxVecs = mean(diffxV01)*1000;
avgDiffyVecs = mean(diffyV01)*1000;
avgDiffzVecs = mean(diffzV01)*1000;
avgDiffxCV0QTM = mean(diffxCV0QTM)*1000;
avgDiffyCV0QTM = mean(diffyCV0QTM)*1000;
avgDiffzCV0QTM = mean(diffzCV0QTM)*1000;
avgDiffxCV1QTM = mean(diffxCV1QTM)*1000;
avgDiffyCV1QTM = mean(diffyCV1QTM)*1000;
avgDiffzCV1QTM = mean(diffzCV1QTM)*1000;
% Present Data
avgDiffVecs = table(avgDiffxVecs, avgDiffyVecs, avgDiffzVecs)
avgDiffQTMv0 = table(avgDiffxCV0QTM, avgDiffyCV0QTM, avgDiffzCV0QTM);
avgDiffQTMv1 = table(avgDiffxCV1QTM, avgDiffyCV1QTM, avgDiffzCV1QTM);

%% Translation Plotting
transPlot = figure(Name="Plot of Translation comparison");

% X plotting
transIndicesX = (xQTM == trans(1));
sgtitle("Position Comparison: transVectors[0] and QTM")
subplot(3,1,1)
plot(time,xCV0, '.r')
hold on
plot(time(transIndicesX),xQTM(transIndicesX), '.k', MarkerSize=1)
ylabel('x [m]')
xlabel('Time [seconds]')
xlim([startTimeTrans stopTimeTrans])
[~, icons] = legend('xCV', 'xQTM', 'Location','eastoutside');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Y plotting - Remember to flip sign, only axis which is aligned
transIndicesY = (yQTM == trans(2));
subplot(3,1,2)
plot(time,yCV0, '.g')
hold on
plot(time(transIndicesY),yQTM(transIndicesY), '.k', MarkerSize=1)
ylabel('y [m]')
xlabel('Time [seconds]')
xlim([startTimeTrans stopTimeTrans])
[~, icons] = legend('yCV0', 'yQTM', 'Location','eastoutside');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

```

```

% Z plotting
transIndicesZ = (zQTM == trans(3));
subplot(3,1,3)
plot(time,zCV0, ...
    '.', 'Color',[109/255, 209/255, 255/255])
hold on
plot(time(transIndicesZ),zQTM(transIndicesZ), '.k', MarkerSize=1)
ylabel('z [m]')
xlabel('Time [seconds]')
xlim([startTimeTrans stopTimeTrans])
[~, icons] = legend('zCV0', 'zQTM', 'Location','eastoutside');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker',...
    'none');
set(icons, 'MarkerSize', 20)

% Export figure
% set(transPlot,'units','normalized','outerposition',[0 0 1 1])
saveas(transPlot, ['poseTest_',num2str(testNr), '_TransPlot'])
saveas(transPlot, ['poseTest_',num2str(testNr), '_TransPlot.png'])

%% Translation Difference between QTM and transVectors[0]
transDiffPlot = figure(Name='TransDiffPlot');
sgtitle('Difference between transVectors[0] and QTM as function of QTM')
dLims = 0.3;

% X plotting
transIndicesX = (xQTM == trans(1));
subplot(3,1,1)
xDiff = xQTM(transIndicesX) - xCV0(transIndicesX);
plot(xQTM(transIndicesX),xDiff, '.r')
ylabel('difference [m]')
xlabel('x [m]')
ylim([-dLims dLims])

% Y plotting - Remember to flip sign, only axis which is aligned
transIndicesY = (yQTM == trans(2));
subplot(3,1,2)
yDiff = yQTM(transIndicesY) - yCV0(transIndicesY);
plot(yQTM(transIndicesY),yDiff, '.g')
ylabel('difference [m]')
xlabel('y [m]')
ylim([-dLims dLims])

% Z plotting
transIndicesZ = (zQTM == trans(3));
subplot(3,1,3)
zDiff = zQTM(transIndicesZ) - zCV0(transIndicesZ);
plot(zQTM(transIndicesZ),zDiff, '.', 'Color',[109/255, 209/255, 255/255])
ylabel('difference [m]')
xlabel('z [m]')
ylim([-dLims dLims])

% Export figure
saveas(transDiffPlot, ['poseTest_',num2str(testNr), '_TransDiffPlot'])
saveas(transDiffPlot, ['poseTest_',num2str(testNr), '_TransDiffPlot.png'])

```

```

%% Rotation Comparison: rotMat0 and QTM
rot0matPlot = figure(Name="Rotation comparison0");
sgtitle("Rotation Matrix Comparison: rotMat0 and QTM")

% Plotting rotVectors[0] with QTM
for i = 1:9
    subplot(3, 3, i);

    vec0 = data(:, i+7); % v0Rij
    QTM = data(:, i+25); % v1Rij

    % Plot only when QTM is tracking
    validIndices = (QTM ~= 0);
    if any(i == [1,4,7])
        plot(time(validIndices), vec0(validIndices), '.r')
    elseif any(i == [2,5,8])
        plot(time(validIndices), vec0(validIndices), '.g')
    else
        plot(time(validIndices), vec0(validIndices), ...
              '.', 'Color',[109/255, 209/255, 255/255])
    end
    hold on
    q = plot(time(validIndices), QTM(validIndices), '.k', MarkerSize=1);

    xlabel('Time [seconds]');
    xlim([startTimeRot stopTimeRot])
    title(['Element ', num2str(i)])
    hold off
    ylim([-1 1])
end

% Add a bit space to the figure
fig = gcf;
fig.Position(3) = fig.Position(3) + 250;

% Add common legend outside subplots
lgdEntries = {'rVec0x','rVec0y','rVec0z', 'QTM'};
hold on
v1 = plot(nan, nan, '.r');
v2 = plot(nan, nan, '.g');
v3 = plot(nan, nan, '.', 'Color',[109/255, 209/255, 255/255]);
[lgd, icons] = legend([v1 v2 v3 q], lgdEntries);
lgd.Position(1) = 0.01;
lgd.Position(2) = 0.4;

% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Export figure
set(rot0matPlot, 'units', 'normalized', 'outerposition',[0 0 1 1])
saveas(rot0matPlot, ['poseTest_', num2str(testNr), '_rot0matPlot'])
saveas(rot0matPlot, ['poseTest_', num2str(testNr), '_rot0matPlot.png'])

%% Rotation: Difference between rotMat0 and QTM
rot0matDiffPlot = figure(Name="RotationDifference0");
sgtitle("Rotation Matrix Element Difference: rotMat0 - QTM")

```

```

% Plotting rotVectors[0] - QTM
for i = 1:9
    subplot(3, 3, i);

    vec0 = data(:, i+7); % v0Rij
    QTM = data(:, i+25); % v1Rij

    % Plot only when QTM is tracking
    validIndices = (QTM ~= 0);
    plot(time(validIndices), (vec0(validIndices)-QTM(validIndices)), ...
        '.k', MarkerSize=1)

    xlabel('Time [seconds]');
    xlim([startTimeRot stopTimeRot])
    title(['Element ', num2str(i)])
    hold off
    ylim([-1 1])
end

% Export figure
set(rot0matDiffPlot, 'units', 'normalized', 'outerposition', [0 0 1 1])
saveas(rot0matDiffPlot, ['poseTest_', num2str(testNr), '_rot0matDiffPlot'])
saveas(rot0matDiffPlot, ['poseTest_', num2str(testNr), '_rot0matDiffPlot.png'])

%% Rotation Comparison: rotMat1 and QTM
rot1matPlot = figure(Name="Rotation Comparison1");
sgtitle("Rotation Matrix Comparison: rotMat1 and QTM")

% Plotting rotVectors[1] with QTM
for i = 1:9
    subplot(3, 3, i);

    vec1 = data(:, i+16); % v0Rij
    QTM = data(:, i+25); % v1Rij

    % Plot only when QTM is tracking
    validIndices = (QTM ~= 0);
    if any(i == [1,4,7])
        plot(time(validIndices), vec1(validIndices), '.r')
    elseif any(i == [2,5,8])
        plot(time(validIndices), vec1(validIndices), '.g')
    else
        plot(time(validIndices), vec1(validIndices), ...
            '.', 'Color',[109/255, 209/255, 255/255])
    end
    hold on
    q = plot(time(validIndices), QTM(validIndices), '.k', MarkerSize=1);

    xlabel('Time [seconds]');
    xlim([startTimeRot stopTimeRot])
    title(['Element ', num2str(i)])
    hold off
    ylim([-1 1])
end

% Add a bit space to the figure
fig = gcf;
fig.Position(3) = fig.Position(3) + 250;

```

```

% Add common legend outside subplots
lgdEntries = {'rVec0x','rVec0y','rVec0z', 'QTM'};
hold on
v1 = plot(nan, nan, '.r');
v2 = plot(nan, nan, '.g');
v3 = plot(nan, nan, '.', 'Color',[109/255, 209/255, 255/255]);
[lgd, icons] = legend([v1 v2 v3 q], lgdEntries);
lgd.Position(1) = 0.01;
lgd.Position(2) = 0.4;

% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Export figure
set(rot1matPlot, 'units', 'normalized', 'outerposition',[0 0 1 1])
saveas(rot1matPlot, ['poseTest_',num2str(testNr), '_rot1matPlot'])
saveas(rot1matPlot, ['poseTest_',num2str(testNr), '_rot1matPlot.png'])

%% Rotation: Difference between rotMat1 and QTM
rot1matDiffPlot = figure(Name="RotationDifference1");
sctitle("Rotation Matrix Element Difference: rotMat1 - QTM")

% Plotting rotVectors[1] with QTM
for i = 1:9
    subplot(3, 3, i);

    vec1 = data(:, i+16); % v0Rij
    QTM = data(:, i+25); % v1Rij

    % Plot only when QTM is tracking
    validIndices = (QTM ~= 0);
    plot(time(validIndices), (vec1(validIndices)-QTM(validIndices)), ...
        '.k', MarkerSize=1)

    xlabel('Time [seconds]');
    xlim([startTimeRot stopTimeRot])
    title(['Element ', num2str(i)])
    hold off
    ylim([-1 1])
end

% Export figure
set(rot1matDiffPlot, 'units', 'normalized', 'outerposition',[0 0 1 1])
saveas(rot1matDiffPlot, ['poseTest_',num2str(testNr), '_rot1matDiffPlot'])
saveas(rot1matDiffPlot, ['poseTest_',num2str(testNr), '_rot1matDiffPlot.png'])

%% Angle Difference: rotMat to Axis-Angle
angleDiffPlot = figure(Name="AngleDiffPlot");
hold on

% Initialize
angChoice = zeros(length(time), 1);

validIndices = [];

```

```

for i = 1 : length(time)
    % Extract Matrices
    rotMat0 = [data(i,8), data(i,9), data(i,10);
               data(i,11), data(i,12), data(i,13);
               data(i,14), data(i,15), data(i,16)];
    rotMat1 = [data(i,17), data(i,18), data(i,19);
               data(i,20), data(i,21), data(i,22);
               data(i,23), data(i,24), data(i,25)];
    rotMatQTM = [data(i,26), data(i,27), data(i,28);
                  data(i,29), data(i,30), data(i,31);
                  data(i,32), data(i,33), data(i,34)];
    % Calculate Cifference Matrices
    rotMatDiff0 = rotMat0' * rotMatQTM;
    rotMatDiff1 = rotMat1' * rotMatQTM;
    % Convert to Axis-Angle
    rotVecDiff0 = rotm2axang(rotMatDiff0);
    rotVecDiff1 = rotm2axang(rotMatDiff1);
    % Extract angle
    ang0 = rad2deg(rotVecDiff0(4));
    ang1 = rad2deg(rotVecDiff1(4));

    % Plot only if QTM matrix is not all zeros
    if all(rotMatQTM(:) ~= 0)
        validIndices(end+1) = i;
    end

    % Choose closest
    if ang0 < ang1
        angChoice(i) = ang0;
    else
        angChoice(i) = ang1;
    end
end

plot(time(validIndices), angChoice(validIndices), '.k', MarkerSize=1)
title("Difference from Axis-Angle: Closest choice")
ylabel("Angle [degrees]")
xlabel("Time [seconds]")
xlim([startTimeRot stopTimeRot])
ylim([0 90])

% Export figure
saveas(angleDiffPlot, ['poseTest_', num2str(testNr), '_angleDiffPlot'])
saveas(angleDiffPlot, ['poseTest_', num2str(testNr), '_angleDiffPlot.png'])

%% XY Projection - Azimuth Angle Calculations

% Enable/disable modulo wrapping of angle and set limits
wrappingX = false;
wrappingY = false;
xStart = -270;
yStart = -360;

% Initialization
CV0anglesXplot = zeros(length(time), 1);
CV0anglesYplot = zeros(length(time), 1);
CV1anglesXplot = zeros(length(time), 1);
CV1anglesYplot = zeros(length(time), 1);
QTAnglesXplot = zeros(length(time), 1);
QTAnglesYplot = zeros(length(time), 1);

```

```

diffAnglesX = zeros(length(time), 1);
diffAnglesY = zeros(length(time), 1);
CVanglesXchoice = zeros(length(time), 1);
CVanglesYchoice = zeros(length(time), 1);

for i = 1 : length(time)
    % Save iteration's components for rotVectors[0]
    x0_x = data(i,8);
    x0_y = data(i,11);
    y0_x = data(i,9);
    y0_y = data(i,12);

    % Save iteration's components for rotVectors[1]
    x1_x = data(i,17);
    x1_y = data(i,20);
    y1_x = data(i,18);
    y1_y = data(i,21);

    % Save iteration's components for QTM
    xQTM_x = data(i,26);
    xQTM_y = data(i,29);
    yQTM_x = data(i,27);
    yQTM_y = data(i,30);

    % Check angle range
    if wrappingX
        % Trig with modulo wrapping
        CV0anglesXplot(i) = mod(atan2d(x0_y,x0_x), 360);
        CV1anglesXplot(i) = mod(atan2d(x1_y,x1_x), 360);
        QTManglesXplot(i) = mod(atan2d(xQTM_y, xQTM_x), 360);
    else
        % Trig w/o wrapping
        CV0anglesXplot(i) = atan2d(x0_y,x0_x);
        CV1anglesXplot(i) = atan2d(x1_y,x1_x);
        QTManglesXplot(i) = atan2d(xQTM_y, xQTM_x);
    end

    if wrappingY
        % Trig with modulo wrapping
        CV0anglesYplot(i) = mod(atan2d(y0_y,y0_x), 360);
        CV1anglesYplot(i) = mod(atan2d(y1_y,y1_x), 360);
        QTManglesYplot(i) = mod(atan2d(yQTM_y, yQTM_x), 360);
    else
        % Trig w/o wrapping
        CV0anglesYplot(i) = atan2d(y0_y,y0_x);
        CV1anglesYplot(i) = atan2d(y1_y,y1_x);
        QTManglesYplot(i) = atan2d(yQTM_y, yQTM_x);
    end

    diff0AnglesX = QTManglesXplot(i) - CV0anglesXplot(i);
    diff0AnglesY = QTManglesYplot(i) - CV0anglesYplot(i);

    diff1AnglesX = QTManglesXplot(i) - CV1anglesXplot(i);
    diff1AnglesY = QTManglesYplot(i) - CV1anglesYplot(i);

```

```

% Choose closest angle from x-axis
if abs(diff0AnglesX) < abs(diff1AnglesX)
    diffAnglesX(i) = diff0AnglesX;
    CVanglesXchoice(i) = CV0anglesXplot(i);
else
    diffAnglesX(i) = diff1AnglesX;
    CVanglesXchoice(i) = CV1anglesXplot(i);
end
% Choose closest angle from y-axis
if abs(diff0AnglesY) < abs(diff1AnglesY)
    diffAnglesY(i) = diff0AnglesY;
    CVanglesYchoice(i) = CV0anglesYplot(i);
else
    diffAnglesY(i) = diff1AnglesY;
    CVanglesYchoice(i) = CV1anglesYplot(i);
end
end

%% Azimuth Angle Plot: Vec0 and QTM

XYprojVec0 = figure(Name="ProjPlotXY_vec0");
sgtitle("Angle projected in XY-plane: Vec0 and QTM")

%%% Plotting X axis angles %%%
validIndices = (data(:,26) ~= 0) & (data(:,29) ~= 0);
subplot(2,1,1)
plot(time(validIndices), CV0anglesXplot(validIndices), '.r')
hold on
plot(time(validIndices), QTManglesXplot(validIndices), '.k', MarkerSize=1)
ylabel("Angle [degrees]")
xlabel("Time [seconds]")
xlim([startTimeRot stopTimeRot])
ylim([xStart (xStart+360)])

[~, icons] = legend('CV0 $\theta$ from X-axis', 'QTM $\theta$ from ... Y-axis', ...
    'Interpreter', 'latex', 'Location', 'best');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

%%% Plotting Y axis angles %%%
validIndices = (data(:,27) ~= 0) & (data(:,30) ~= 0);
subplot(2,1,2)
plot(time(validIndices), CV0anglesYplot(validIndices), '.g')
hold on
plot(time(validIndices), QTManglesYplot(validIndices), '.k', MarkerSize=1)
ylabel("Angle [degrees]")
xlabel("Time [seconds]")
xlim([startTimeRot stopTimeRot])
ylim([yStart (yStart+360)])

[~, icons] = legend('CV0 $\theta$ from Y-axis', 'QTM $\theta$ from ... Y-axis', ...
    'Interpreter', 'latex', 'Location', 'best');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

```

```

% Export figure
%set(rot1matDiffPlot,'units','normalized','outerposition',[0 0 1 1])
saveas(XYprojVec0, ['poseTest_', num2str(testNr), '_XYprojVec0'])
saveas(XYprojVec0, ['poseTest_', num2str(testNr), '_XYprojVec0.png'])

%% Azimuth Angle Plot: Vec1 and QTM

% Plot only when QTM rotation matrix != 0
XYprojVec1 = figure(Name="ProjPlotXY_vec1");
sgtitle("Angle projected in XY-plane: Vec1 and QTM")

%%% Plotting X axis angles %%%
validIndices = (data(:,26) ~= 0) & (data(:,29) ~= 0);
subplot(2,1,1)
plot(time(validIndices), CV1anglesXplot(validIndices), '.r')
hold on
plot(time(validIndices), QTMManglesXplot(validIndices), '.k', MarkerSize=1)
ylabel("Angle [degrees]")
xlabel("Time [seconds]")
ylim([xStart (xStart+360)])
xlim([startTimeRot stopTimeRot])

[~, icons] = legend('CV1 $\theta$ from X-axis', 'QTM $\theta$ from ... Y-axis', ...
    'Interpreter', 'latex', 'Location', 'best');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

%%% Plotting Y axis angles %%%
validIndices = (data(:,27) ~= 0) & (data(:,30) ~= 0);
subplot(2,1,2)
plot(time(validIndices), CV1anglesYplot(validIndices), '.g')
hold on
plot(time(validIndices), QTMManglesYplot(validIndices), '.k', MarkerSize=1)
ylabel("Angle [degrees]")
xlabel("Time [seconds]")
ylim([yStart (yStart+360)])
xlim([startTimeRot stopTimeRot])

[~, icons] = legend('CV1 $\theta$ from Y-axis', 'QTM $\theta$ from ... Y-axis', ...
    'Interpreter', 'latex', 'Location', 'best');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

```

```

% Export figure
%set(rot1matDiffPlot,'units','normalized','outerposition',[0 0 1 1])
saveas(XYprojVec1, ['poseTest_',num2str(testNr), '_XYprojVec1'])
saveas(XYprojVec1, ['poseTest_',num2str(testNr), '_XYprojVec1.png'])

%% Azimuth Angle Plot: Vector Choice and QTM

XYprojVecChoice = figure(Name="ProjPlotXY_vecChoice");
sgtitle("Angle projected in XY-plane: Closest rotVector and QTM")

%%% Plotting X axis angles %%%
validIndices = (data(:,26) ~= 0) & (data(:,29) ~= 0);
subplot(2,1,1)
plot(time(validIndices), CVanglesXchoice(validIndices), '.r')
hold on
plot(time(validIndices), QTManglesXplot(validIndices), '.k', MarkerSize=1)
ylabel("Angle [degrees]")
xlabel("Time [seconds]")
ylim([xStart (xStart+360)])
xlim([startTimeRot stopTimeRot])

[~, icons] = legend('CV $\theta$ from X-axis', 'QTM $\theta$ from Y-axis', ...
    'Interpreter', 'latex', 'Location','best');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

%%% Plotting Y axis angles %%%
validIndices = (data(:,27) ~= 0) & (data(:,30) ~= 0);
subplot(2,1,2)
plot(time(validIndices), CVanglesYchoice(validIndices), '.g')
hold on
plot(time(validIndices), QTManglesYplot(validIndices), '.k', MarkerSize=1)
ylabel("Angle [degrees]")
xlabel("Time [seconds]")
ylim([yStart (yStart+360)])
xlim([startTimeRot stopTimeRot])

[~, icons] = legend('CV $\theta$ from Y-axis', 'QTM $\theta$ from Y-axis', ...
    'Interpreter', 'latex', 'Location','best');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Export figure
%set(rot1matDiffPlot,'units','normalized','outerposition',[0 0 1 1])
saveas(XYprojVecChoice, ['poseTest_',num2str(testNr), '_XYprojVecChoice'])
saveas(XYprojVecChoice, ['poseTest_',num2str(testNr), '_XYprojVecChoice.png'])

%% Difference Azimuth Angle Plot: Choice - QTM

validIndices = (data(:,26) ~= 0) & (data(:,27) ~= 0) ...
    & (data(:,29) ~= 0) & (data(:,30) ~= 0);
XYprojDiff = figure(Name="ProjPlotXY");
plot(time(validIndices), diffAnglesX(validIndices), '.r', MarkerSize=1)
hold on
plot(time(validIndices), diffAnglesY(validIndices), '.g', MarkerSize=1)
title("Difference in angle projected in XY-plane: Choice")

```

```

ylabel("Angle [degrees]")
xlabel("Time [seconds]")
[h, icons] = legend('X-axis diff', 'Y-axis diff');
ylim([-30 30])
xlim([startTimeRot stopTimeRot])

% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Export figure
%set(rot1matDiffPlot,'units','normalized','outerposition',[0 0 1 1])
saveas(XYprojDiff, ['poseTest_', num2str(testNr), '_XYprojDiff'])
saveas(XYprojDiff, ['poseTest_', num2str(testNr), '_XYprojDiff.png'])

```

### G.2.7 distributionFitting.m

```

clc; clear; close all;
%% Import, Append and Plot Data - Translation

for testNr = 0 : 4
    if (testNr ~= 3)
        fileName = ['ExportedResults_', num2str(testNr), '.csv'];
        data = csvread(fileName, 1,0);
        if testNr == 0
            time = data(:,1);
            xQTM = data(:,2);
            yQTM = data(:,3);
            zQTM = data(:,4);
            xCV0 = data(:,5);
            yCV0 = -data(:,6); % Flip axis, different frames
            zCV0 = data(:,7);
        else
            time = [time; data(:,1)];
            xQTM = [xQTM; data(:,2)];
            yQTM = [yQTM; data(:,3)];
            zQTM = [zQTM; data(:,4)];
            xCV0 = [xCV0; data(:,5)];
            yCV0 = [yCV0; -data(:,6)]; % Flip axis, different frames
            zCV0 = [zCV0; data(:,7)];
        end
    end
end
trans = [0.190 0.143 1.564]; % Physically measured in m

% Size of dots in transPlot
tSize = 1;

transDiffPlot = figure(Name='TransDiffPlot2');
sgtitle('Difference between transVector[0] and QTM as function of QTM')
xLimY1 = -0.07;
xLimY2 = 0.12;
yLimY1 = -0.06;
yLimY2 = 0.11;
zLimY1 = -0.25;
zLimY2 = 0.28;

% X plotting
transIndicesX = (xQTM ~= trans(1));
subplot(3,1,1)
xDiff = xQTM(transIndicesX) - xCV0(transIndicesX);
plot(xQTM(transIndicesX),xDiff, '.r', MarkerSize=tSize)
ylabel('difference [m]')
xlabel('x [m]')
xlim([-0.5 0.6])
ylim([xLimY1 xLimY2])

% Y plotting - Sign is flipped above, only axis which is aligned
transIndicesY = (yQTM ~= trans(2));
subplot(3,1,2)
yDiff = yQTM(transIndicesY) - yCV0(transIndicesY);
plot(yQTM(transIndicesY),yDiff, '.g', MarkerSize=tSize)
ylabel('difference [m]')
xlabel('y [m]')
ylim([yLimY1 yLimY2])

```

```

% Z plotting
transIndicesZ = (zQTM == trans(3));
subplot(3,1,3)
zDiff = zQTM(transIndicesZ) - zCV0(transIndicesZ);
plot(zQTM(transIndicesZ),zDiff, '.', 'Color', ...
[109/255, 209/255, 255/255], MarkerSize=tSize)
ylabel('difference [m]')
xlabel('z [m]')
xlim([0.1 1.6])
ylim([zLimY1 zLimY2])

% Variables for distributions
distDotSize = 1; % Size of plotted points
boxTrans = 0.07; % Color of boxes outside \pm 1*sigma

% Export figure
saveas(transDiffPlot, 'transDiffPlotCombined')

%% Z Distributions

zQTM = zQTM(transIndicesZ);

% Sort the lists to ascending order
[zQTMsorted, zQTMindices] = sort(zQTM);
zDiffSorted = zDiff(zQTMindices);

% Split the data into evenly distributed intervals
distNums = 9;
intervals = linspace(zQTMsorted(1), zQTMsorted(end), (distNums+1));

% Find indices closest to bounds of intervals
[~, closestIdx] = min( abs(zQTMsorted - intervals));

% Allocate space for distribution lists
zStdDev = zeros(distNums, 1);
zExpVal = zeros(distNums, 1);
zIntervalMid = zeros(distNums, 1);

zIntervalPlots = figure(Name="zIntervalPlots");
sgtitle('Interval Plots of z-axis')

for i = 1 : distNums
    % Save current interval values
    curQTM = zQTMsorted( closestIdx(i):closestIdx(i+1) );
    curDiff = zDiffSorted( closestIdx(i):closestIdx(i+1) );

    % Find horizontal bounds
    left = min(curQTM);
    right = max(curQTM);

    % Plot intervals
    subplot(sqrt(distNums),sqrt(distNums),i)
    plot(curQTM, curDiff, '.', 'Color',[109/255, 209/255, 255/255], ...
        MarkerSize=distDotSize)
    ylabel('difference [m]')
    xlabel('z [m]')
    xlim([left right])
    ylim([zLimY1 zLimY2])
    midVal = mean(xlim);
    zIntervalMid(i) = midVal;

```

```

% Scale for distribution plotting
xLims = xlim;
distScale = ( xLims(2) - xLims(1) )/3;

% Fit Normal Distribution
zPD = fitdist(curDiff, 'normal');
zStdDev(i) = zPD.sigma;
zExpVal(i) = zPD.mu;

% Plot Distribution
hold on
yline(zExpVal(i)+zStdDev(i), 'k')
yline(zPD.mu-zPD.sigma, 'k')
% Grey area below
patch([left, right, right, left], ...
[zLimY1, zLimY1, zPD.mu-zPD.sigma, zPD.mu-zPD.sigma], ...
'k', 'FaceAlpha', boxTrans, 'EdgeColor', 'none')
% Grey area above
patch([left, right, right, left], ...
[zPD.mu+zPD.sigma, zPD.mu+zPD.sigma, zLimY2, zLimY2], ...
'k', 'FaceAlpha', boxTrans, 'EdgeColor', 'none')
end

% Export figure
set(zIntervalPlots,'units','normalized','outerposition',[0 0 1 1])
saveas(zIntervalPlots, 'zIntervalPlots')

%% X Distributions

xQTM = xQTM(transIndicesX);

% Sort the lists to ascending order
[xQTMsorted, xQTMindices] = sort(xQTM);
xDiffSorted = xDiff(xQTMindices);

% Split the data into evenly distributed intervals
distNums = 9;
intervals = linspace(xQTMsorted(1), xQTMsorted(end), (distNums+1));

% Find indices closest to bounds of intervals
[~, closestIdx] = min( abs(xQTMsorted - intervals) );

% Allocate space for distribution lists
xStdDev = zeros(distNums, 1);
xExpVal = zeros(distNums, 1);
xIntervalMid = zeros(distNums, 1);

xIntervalPlots = figure(Name="xIntervalPlots");
sgtitle('Interval Plots of x-axis')

for i = 1 : distNums
    % Save current interval values
    curQTM = xQTMsorted( closestIdx(i):closestIdx(i+1) );
    curDiff = xDiffSorted( closestIdx(i):closestIdx(i+1) );

    % Find horizontal bounds
    left = min(curQTM);
    right = max(curQTM);

```

```

% Plot intervals
subplot(sqrt(distNums), sqrt(distNums), i)
plot(curQTM, curDiff, '.r', MarkerSize=distDotSize)
ylabel('difference [m]')
xlabel('x [m]')
xlim([left right])
ylim([xLimY1 xLimY2])
midVal = mean(xlim);
xIntervalMid(i) = midVal;

% Scale for distribution plotting
xLims = xlim;
distScale = ( xLims(2) - xLims(1) )/3;

% Fit Normal Distribution
xPD = fitdist(curDiff, 'normal');
xStdDev(i) = xPD.sigma;
xExpVal(i) = xPD.mu;

% Plot Distribution
hold on
yline(xExpVal(i)+xStdDev(i), 'k')
yline(xPD.mu-xPD.sigma, 'k')
% Grey area below
patch([left, right, right, left], ...
       [xLimY1, xLimY1, xPD.mu-xPD.sigma, xPD.mu-xPD.sigma], ...
       'k', 'FaceAlpha', boxTrans, 'EdgeColor', 'none')
% Grey area above
patch([left, right, right, left], ...
       [xPD.mu+xPD.sigma, xPD.mu+xPD.sigma, xLimY2, xLimY2], ...
       'k', 'FaceAlpha', boxTrans, 'EdgeColor', 'none')
end

% Export figure
set(xIntervalPlots, 'units', 'normalized', 'outerposition', [0 0 1 1])
saveas(xIntervalPlots, 'xIntervalPlots')

%% Y Distributions

yQTM = yQTM(transIndicesX);

% Sort the lists to ascending order
[yQTMsorted, yQTMindices] = sort(yQTM);
yDiffSorted = yDiff(yQTMindices);

% Split the data into evenly distributed intervals
distNums = 9;
intervals = linspace(yQTMsorted(1), yQTMsorted(end), (distNums+1));

% Find indices closest to bounds of intervals
[~, closestIdx] = min(abs(yQTMsorted - intervals));

% Allocate space for distribution lists
yStdDev = zeros(distNums, 1);
yExpVal = zeros(distNums, 1);
yIntervalMid = zeros(distNums, 1);

yIntervalPlots = figure(Name="yIntervalPlots");
sgtitle('Interval Plots of y-axis')

```

```

for i = 1 : distNums
    % Save current interval values
    curQTM = yQTMsorted( closestIdx(i):closestIdx(i+1) );
    curDiff = yDiffSorted( closestIdx(i):closestIdx(i+1) );

    % Find horizontal bounds
    left = min(curQTM);
    right = max(curQTM);

    % Plot intervals
    subplot(sqrt(distNums),sqrt(distNums),i)
    plot(curQTM, curDiff, '.g', MarkerSize=distDotSize)
    ylabel('difference [m]')
    xlabel('y [m]')
    xlim([left, right])
    ylim([yLimY1 yLimY2])
    midVal = mean(xlim);
    yIntervalMid(i) = midVal;

    % Scale for distribution plotting
    xLims = xlim;
    distScale = ( xLims(2) - xLims(1) )/3;

    % Fit Normal Distribution
    yPD = fitdist(curDiff, 'normal');
    yStdDev(i) = yPD.sigma;
    yExpVal(i) = yPD.mu;

    % Plot Distribution
    hold on
    yline(yExpVal(i)+yStdDev(i), 'k')
    yline(yPD.mu-yPD.sigma, 'k')
    % Grey area below
    patch([left, right, right, left], ...
        [yLimY1, yLimY1, yPD.mu-yPD.sigma, yPD.mu-yPD.sigma], ...
        'k', 'FaceAlpha', boxTrans, 'EdgeColor', 'none')
    % Grey area above
    patch([left, right, right, left], ...
        [yPD.mu+yPD.sigma, yPD.mu+yPD.sigma, yLimY2, yLimY2], ...
        'k', 'FaceAlpha', boxTrans, 'EdgeColor', 'none')
end

% Export figure
set(yIntervalPlots,'units','normalized','outerposition',[0 0 1 1])
saveas(yIntervalPlots, 'yIntervalPlots')

```

### G.2.8 KalmanFilter1D.m

```

clc; clear; close all;

%% Implement data
testNr = 4;
fileName = ['ExportedResults_', num2str(testNr), '.csv'];
data = csvread(fileName, 1,0);

time = data(:,1);
xQTM = data(:,2);
yQTM = data(:,3);
zQTM = data(:,4);
xCV0 = data(:,5);
yCV0 = -data(:,6); % Flip axis, different frames
zCV0 = data(:,7);

trans = [0.190 0.143 1.564]; % Physically measured in m

% xlims: position and speed
startTime = 0;
stopTime = time(end);

%% Interpolation

% New timeseries with constant period
Hz = 100; % [samples/second]
dt = 1/Hz; % [seconds]
t = 0 : dt : time(end); % [seconds]

% Change no detection to NaN
nanIdx = find(zQTM == trans(3));
zQTMi = zQTM;
zQTMi(nanIdx) = NaN;

% Interpolation: (oldTime, interpBetween, newTime)
zQTMi = interp1(time(~isnan(zQTMi)), zQTMi(~isnan(zQTMi)), t);

% Remove NaN's after data
while isnan(zQTMi(end))
    zQTMi(end) = [];
    t(end) = [];
end

% Interpolation Plotting
interpPlot = figure(Name="zQTMinterpolation");
plot(t, zQTMi, '.', Color=[.85 .85 .85])
hold on
validIndicesQTM = (zQTM ~= trans(3));
plot(time(validIndicesQTM), zQTM(validIndicesQTM), '.', ...
      'Color', '#21007F', MarkerSize=1)
title('Interpolation of QTM data')
[~, icons] = legend('InterpolatedQTM', 'QTM', 'Location', 'southwest');
xlim([startTime stopTime])
ylabel('Meters')
xlabel('Seconds')

```

```

% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Export figure
set(interpPlot, 'units', 'normalized', 'outerposition', [0 0 1 1])
saveas(interpPlot, ['poseTest_', num2str(testNr), '_interpolation'])
saveas(interpPlot, ['poseTest_', num2str(testNr), '_interpolation.png'])

%% Speed and Acceleration

% Preallocate Space
zDotQTM = zeros(length(zQTMi)-1,1);
zDotDotQTM = zeros(length(zQTMi)-1,1);

% Save Recorded Speed z'
for i = 2 : length(t)
    zDotQTM(i) = (zQTMi(i) - zQTMi(i-1))/dt; % [m/s]
end

% Save Recorded Acceleration z ''
for i = 2 : length(t)
    zDotDotQTM(i) = (zDotQTM(i) - zDotQTM(i-1))/(dt); % [m/s^2]
end

% Set default seed for consistent simulation noise
rng("default")

% Output Noise Density - Datasheet BMI088
ond = 190*10^-6;      % [g/sqrt(Hz)]
g = 9.80665;          % [m/s^2]
% Convert to scalar
scale = ond*sqrt(Hz)/g;
% Zero Mean White Gaussian Noise
Wn = wgn(length(zDotDotQTM),1,scale, 'linear');

% Noisy IMU signal
simIMU = zDotDotQTM + Wn;
% Standard Deviation for Covariance (matrix) Q
stdDevIMU = std(Wn);

%% Kalman Filter

% Initialization
timeTol = 0.01;        % Sync low sps to high sps
x = [ zQTM(1); zDotQTM(1)]; % Start [z, zDot]

% Preallocate Space
zKalman = zeros(length(t),1);
zDotKalman = zeros(length(t),1);

% State Transition Matrix
A = [ 1 dt ;
      0 1 ];
% Control Input Matrix
B = [ dt^2/2 ;
      dt ];
% Measurement Matrix
C = [ 1 0 ];

```

```

% Identity Matrix
I = eye(2);

% Process Noise w_n
Q = cov(Wn); % Covariance (Matrix)
Q2 = stdDevIMU^2;
Q3 = var(Wn);

% zSigma import: f(z) = p1*z + p2
p1 = 0.0380;
p2 = -0.0029;

% Measurement Noise v_n
% R = 0.15; % Covariance (Matrix)

% State Covariance (Matrix)
P = B*Q*B';

% Constant Rate: Kalman Filter
for i = 2 : length(t)

    % Find closest value
    for CViter = 1 : length(time)
        if (abs(time(CViter) - t(i)) < timeTol)
            y = zCV0(CViter);
            % detected = true;
            break
        else
            y = NaN;
            % detected = false;
        end
    end

    % Update Input: Simulated Acceleration
    u = simIMU(i);

    % Update Measurement Noise
    R = (p1*y + p2)^2;

    % Prediction
    x = A*x + B*u;
    P = A*P*A' + B*Q*B';

    % Measurement Update
    if ~isnan(y) % if detected
        Kk = (P*C')/(C*P*C' + R);
        x = x + Kk*(y - C*x);
        P = (I - Kk*C)*P;
    end

    % Extract Filtered Value of position
    zKalman(i) = x(1);
    zDotKalman(i) = x(2);
end

```

```

% Calculate differences
diff = [];
idxDiff = [];
for i = 1 : length(t)
    if t(i) > stopTime
        break
    end
    if ~isnan(zQTMi(i)) && (t(i) > startTime)
        diff(end+1) = abs(zKalman(i) - zQTMi(i));
        idxDiff(end+1) = i;
    end
end

meas = [];
idxMeas = [];
for i = 1 : length(time)
    if time(i) > stopTime
        break
    end
    if (zQTM(i) ~= trans(3)) && (time(i) > startTime)
        meas(end+1) = abs(zCV0(i) - zQTM(i));
        idxMeas(end+1) = i;
    end
end

% Extract and display statistics - [mm]
    % Difference
avgDiff = mean((diff))*1000;
stdDiff = std((diff))*1000;
table(avgDiff, stdDiff)
    % Measurements
avgMeas = mean((meas))*1000;
stdMeas = std((meas))*1000;
table(avgMeas, stdMeas)

%% Translation Plotting
transPlot = figure(Name="TranslationPlot");

% Plot Measured Position from D435 Camera
plot(time,zCV0, ...
    '.', 'Color',[109/255, 209/255, 255/255])
hold on
% Plot Reference from QTM
plot(t,zQTMi, '.k', MarkerSize=1)
% Plot Kalman Filter Estimate Position
plot(t, zKalman, '.b', MarkerSize=1)

[~, icons] = legend('zCV0', 'zQTMinterp', 'zKalman', 'Location', 'southwest');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

ylabel('Meters')
xlabel('Seconds')
xlim([startTime stopTime])
title('1D Kalman Filter')

```

```

set(transPlot,'units','normalized','outerposition',[0 0 1 1])
% Export figure
saveas(transPlot, ['poseTest_',num2str(testNr), '_transKalmanPlot'])
saveas(transPlot, ['poseTest_',num2str(testNr), '_transKalmanPlot.png'])

%% Difference Plotting
% Transpose for plotting
diff = diff';
meas = meas';

% Plot Differences between filter and no filter
filterNoFilter1D = figure(Name='FilterNoFilter1D');
measurements = plot(time(idxMeas),meas,'.', ...
    'Color',[109/255, 209/255, 255/255]);
hold on
filter = plot(t(idxDiff),diff,'.b', MarkerSize=1);
title('Comparison: abs(error) of measurement and filter')

% hold on
diffSize = 1.5;
diffColor = '#0000B5';
measColor = '#7EBBD6';
% yline((avgMeas)/1000,'-','Color',measColor,'LineWidth',diffSize)
yline((avgMeas+stdMeas)/1000,'--','Color',measColor,'LineWidth',diffSize)
yline((avgMeas-stdMeas)/1000,'--','Color',measColor,'LineWidth',diffSize)
% yline((avgDiff)/1000,'-','Color', diffColor, 'LineWidth',diffSize)
yline((avgDiff+stdDiff)/1000,'--','Color', diffColor,'LineWidth',diffSize)
yline((avgDiff-stdDiff)/1000,'--','Color', diffColor,'LineWidth',diffSize)

% Plot NaNs for legend
measLine = plot(nan, nan, '--', 'Color', measColor);
diffLine = plot(nan, nan, '--', 'Color', diffColor);

ylabel('Meters')
xlabel('Seconds')
[~, icons] = legend([measurements filter measLine diffLine], ...
    'zCV0-zQTM', 'Kalman-zQTMi', 'Measurement Confidence Interval', ...
    'Filter Confidence Interval', 'Location','northeast');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)
xlim([startTime stopTime])

set(filterNoFilter1D,'units','normalized','outerposition',[0 0 1 1])
% Export figure
saveas(filterNoFilter1D, ['poseTest_',num2str(testNr), '_filterNoFilter1D'])
saveas(filterNoFilter1D, ['poseTest_',num2str(testNr), ...
    '_filterNoFilter1D.png'])

%% Speed Plotting
speedPlot = figure(Name="SpeedPlot");

% Plot Reference Speed
plot(t,zDotQTM, '.k')
hold on

```

```

% Plot Estimated Speed
plot(t,zDotKalman, '.', 'Color', '#B1A9D8', MarkerSize=1)

[~, icons] = legend('zDotQTM', 'zDotKalman', 'Location','northeast');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

ylabel('zDot [m/s]')
xlabel('Time [seconds]')
xlim([startTime stopTime])
title("Speed: Kalman' and QTM'")

set(speedPlot,'units','normalized','outerposition',[0 0 1 1])

%% Acceleration Plotting
accPlot = figure(Name="AccPlot");
% Plot QTM Acceleration
plot(t,zDotDotQTM, '.k')
hold on
% Plot Simulation with Noise
plot(t,simIMU, '.', 'Color', '#F7B9EA', MarkerSize=1)

[a, icons] = legend('IMUsim', 'zDotDotQTM', 'Location','northeast');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

ylabel('zDotDot [m/s^2]')
xlabel('Time [seconds]')
xlim([startTime stopTime])
title("Acceleration: simIMU and QTM'")

set(accPlot,'units','normalized','outerposition',[0 0 1 1])

```

### G.2.9 KalmanFilter3D.m

```

clc; clear; close all;

%% Implement data
testNr = 4;
fileName = ['ExportedResults_', num2str(testNr), '.csv'];
data = csvread(fileName, 1,0);

time = data(:,1);
xQTM = data(:,2);
yQTM = data(:,3);
zQTM = data(:,4);
xCV0 = data(:,5);
yCV0 = -data(:,6); % Flip axis, different frames
zCV0 = data(:,7);

trans = [0.190 0.143 1.564]; % Physically measured in m

% Markersizes
mSize = 4;
qSize = 3;
kSize = 3.5;
iSize = 6;
rSize = 4;

% xlims: position and speed
startTime = 0;
stopTime = time(end);

%% Interpolation

% New timeseries with constant period
Hz = 100; % [samples/second]
dt = 1/Hz; % [seconds]
t = 0 : dt : time(end); % [seconds]

% Change no detection to NaN
xNanIdx = find(xQTM == trans(1));
yNanIdx = find(yQTM == trans(2));
zNanIdx = find(zQTM == trans(3));
xQTMi = xQTM;
xQTMi(xNanIdx) = NaN;
yQTMi = yQTM;
yQTMi(yNanIdx) = NaN;
zQTMi = zQTM;
zQTMi(zNanIdx) = NaN;

% Interpolation: (oldTime, interpBetween, newTime)
xQTMi = interp1(time(~isnan(xQTMi)), xQTMi(~isnan(xQTMi)), t);
yQTMi = interp1(time(~isnan(yQTMi)), yQTMi(~isnan(yQTMi)), t);
zQTMi = interp1(time(~isnan(zQTMi)), zQTMi(~isnan(zQTMi)), t);

% Remove last NaN's - same indices for all
while isnan(zQTMi(end))
    xQTMi(end) = [];
    yQTMi(end) = [];
    zQTMi(end) = [];
    t(end) = [];
end

```

```

% Plotting
    % X
xInterp = figure(Name="xQTMinterpolation");
plot(t, xQTMi,'.', Color=[.85 .85 .85], MarkerSize=iSize)
hold on
validIndicesQTMx = (xQTM ~= trans(1));
plot(time(validIndicesQTMx),xQTM(validIndicesQTMx), ...
    '.', 'Color','#9E0000',MarkerSize=rSize)
title('Interpolation of QTM data: x')
[~, icons] = legend('xInterpolatedQTM', 'xQTM', 'Location', 'northeast');
xlim([startTime stopTime])
ylabel('Meters')
xlabel('Seconds')

% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Save Figure
set(xInterp,'units','normalized','outerposition',[0 0 1 1])
saveas(xInterp, ['poseTest_',num2str(testNr), '_xInterpolation3D'])
saveas(xInterp, ['poseTest_',num2str(testNr), '_xInterpolation3D.png'])

    % Y
yInterp = figure(Name="yQTMinterpolation");
plot(t, yQTMi,'.', Color=[.85 .85 .85], MarkerSize=iSize)
hold on
validIndicesQTMy = (yQTM ~= trans(2));
plot(time(validIndicesQTMy),yQTM(validIndicesQTMy), ...
    '.', 'Color','#176D00', MarkerSize=rSize)
title('Interpolation of QTM data: y')
[~, icons] = legend('yInterpolatedQTM', 'yQTM', 'Location', 'northeast');
xlim([startTime stopTime])
ylabel('Meters')
xlabel('Seconds')

% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Save Figure
set(yInterp,'units','normalized','outerposition',[0 0 1 1])
saveas(yInterp, ['poseTest_',num2str(testNr), '_yInterpolation3D'])
saveas(yInterp, ['poseTest_',num2str(testNr), '_yInterpolation3D.png'])

    % Z
zInterp = figure(Name="zQTMinterpolation");
plot(t, zQTMi,'.', Color=[.85 .85 .85], MarkerSize=iSize)
hold on
validIndicesQTMz = (zQTM ~= trans(3));
plot(time(validIndicesQTMz),zQTM(validIndicesQTMz), ...
    '.', 'Color','#21007F', MarkerSize=rSize)
title('Interpolation of QTM data: z')
[~, icons] = legend('zInterpolatedQTM', 'zQTM', 'Location', 'southwest');
xlim([startTime stopTime])
ylabel('Meters')
xlabel('Seconds')

```

```

% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

% Save Figure
set(zInterp,'units','normalized','outerposition',[0 0 1 1])
saveas(zInterp, ['poseTest_',num2str(testNr), '_zInterpolation3D'])
saveas(zInterp, ['poseTest_',num2str(testNr), '_zInterpolation3D.png'])

%% Speed and Acceleration

% Preallocate Space
xDotQTM = zeros(length(xQTMi)-1,1);
xDotDotQTM = zeros(length(xQTMi)-1,1);
yDotQTM = zeros(length(yQTMi)-1,1);
yDotDotQTM = zeros(length(yQTMi)-1,1);
zDotQTM = zeros(length(zQTMi)-1,1);
zDotDotQTM = zeros(length(zQTMi)-1,1);

% Save Recorded Speeds
for i = 2 : length(t)
    xDotQTM(i) = (xQTMi(i) - xQTMi(i-1))/dt;
    yDotQTM(i) = (yQTMi(i) - yQTMi(i-1))/dt;
    zDotQTM(i) = (zQTMi(i) - zQTMi(i-1))/dt;
end

% Save Recorded Accelerations
for i = 2 : length(t)
    xDotDotQTM(i) = (xDotQTM(i) - xDotQTM(i-1))/(dt); % [m/s^2]
    yDotDotQTM(i) = (yDotQTM(i) - yDotQTM(i-1))/(dt); % [m/s^2]
    zDotDotQTM(i) = (zDotQTM(i) - zDotQTM(i-1))/(dt); % [m/s^2]
end

% Set default seed for consistent simulation noise
rng("default")

% Output Noise Density - Datasheet BMI088
xyOND = 160*10^-6;          % [ug/sqrt(Hz)]
zOND = 190*10^-6;          % [ug/sqrt(Hz)]
g = 9.80665;               % [m/s^2]
% Convert to scalar
xyScale = xyOND*sqrt(Hz)/g;
zScale = zOND*sqrt(Hz)/g;
% Zero Mean White Gaussian Noise - One for each to get different noise
xWn = wgn(length(xDotDotQTM),1,xyScale, 'linear');
yWn = wgn(length(yDotDotQTM),1,xyScale, 'linear');
zWn = wgn(length(zDotDotQTM),1,zScale, 'linear');

% Noisy IMU signal
xSimIMU = xDotDotQTM + zWn;
ySimIMU = yDotDotQTM + zWn;
zSimIMU = zDotDotQTM + zWn;

%% 3D Kalman Filter

% Initialization
timeTol = 0.01;           % Sync low sps to high sps

```

```

% State Vector 6x1
x = [ xQTM(1);
       xDotQTM(1);
       yQTM(1);
       yDotQTM(1);
       zQTM(1);
       zDotQTM(1) ];

% Preallocate Space
xKalman = zeros(length(t),1);
xDotKalman = zeros(length(t),1);
yKalman = zeros(length(t),1);
yDotKalman = zeros(length(t),1);
zKalman = zeros(length(t),1);
zDotKalman = zeros(length(t),1);

% State Transition Matrix 6x6
A = [ 1 dt 0 0 0 0 ;
       0 1 0 0 0 0 ;
       0 0 1 dt 0 0 ;
       0 0 0 1 0 0 ;
       0 0 0 0 1 dt ;
       0 0 0 0 0 1 ];
% Control Input Matrix: 6x3
B = [ dt^2/2 0 0 ; % x'
       dt 0 0 ; % x'
       0 dt^2/2 0 ; % y'
       0 dt 0 ; % y'
       0 0 dt^2/2 ; % z'
       0 0 dt ]; % z'

% Measurement Matrix 3x6
C = [ 1 0 0 0 0 0 ; % x
       0 0 1 0 0 0 ; % y
       0 0 0 0 1 0 ]; % z

% Identity Matrix
I = eye(6);

% Process Noise w_n 3x3
Q = [ std(xWn)^2 0 0 ;
       0 std(yWn)^2 0 ;
       0 0 std(zWn)^2 ];

% xSigma import: f(x) = p1*x^2 + p2*x + p3
p1x = 0.0330;
p2x = -0.0039;
p3x = 0.0098;
% ySigma import: f(y) = p1*y^2 + p2*y + p3 - 9 intervals
p1y = 0.0419;
p2y = -0.0014;
p3y = 0.0110;
% ySigma import: f(y) = p1*y^2 + p2*y + p3 - 16 intervals
% p1y = 0.0363;
% p2y = -0.0026;
% p3y = 0.0112;
% zSigma import: f(z) = p1*z + p2
p1z = 0.0380;
p2z = -0.0029;

```

```

% Start Measurement Noise
v = [ (p1x*xCV0(1)^2 + p2x*xCV0(1) + p3x)^2 ;  

      (p1y*yCV0(1)^2 + p2y*yCV0(1) + p3y)^2 ;  

      (p1z*zCV0(1) + p2z)^2 ];

% State Covariance (Matrix)
P = B*Q*B';

% Constant Rate: Kalman Filter
for i = 2 : length(t)

    % Find closest value
    for CViter = 1 : length(time)
        if (abs(time(CViter) - t(i)) < timeTol)
            y = [xCV0(CViter); yCV0(CViter); zCV0(CViter)];

            % Update Measurement Noise
            v = [ (p1x*x(1)^2 + p2x*x(1) + p3x)^2 ; % x  

                  (p1y*x(3)^2 + p2y*x(3) + p3y)^2 ; % y  

                  (p1z*x(5) + p2z)^2 ]; % z
            R = diag(v); % + 1e-6*I;
            % detected = true;
            break
        else
            y = NaN;
            % detected = false;
        end
    end

    % Update Input: Simulated Acceleration
    u = [xSimIMU(i); ySimIMU(i); zSimIMU(i)];

    % Prediction
    x = A*x + B*u;
    P = A*P*A' + B*Q*B';

    if ~isnan(y) % if detected

        % Measurement Update
        Kk = (P*C')/(C*P*C' + R);
        x = x + Kk*(y - C*x);
        P = (I - Kk*C)*P;
    end

    % Extract Filtered Value of position
    xKalman(i)      = x(1);
    xDotKalman(i)   = x(2);
    yKalman(i)      = x(3);
    yDotKalman(i)   = x(4);
    zKalman(i)      = x(5);
    zDotKalman(i)   = x(6);
end

% Calculate differences
xFilter = [];
yFilter = [];
idxXf= [];
idxYf= [];

```

```

for i = 1 : length(t)
    if t(i) > stopTime
        break
    end
    if ~isnan(xQTMi(i)) && (t(i) > startTime)
        xFilter(end+1) = abs(xKalman(i) - xQTMi(i));
        idxXf(end+1) = i;
    end
    if ~isnan(yQTMi(i)) && (t(i) > startTime)
        yFilter(end+1) = abs(yKalman(i) - yQTMi(i));
        idxYf(end+1) = i;
    end
end

xMeas = [];
yMeas = [];
idxXMeas = [];
idxYMeas = [];
for i = 1 : length(time)
    if time(i) > stopTime
        break
    end
    if (xQTM(i) ~= trans(1)) && (xCV0(i) ~= 0) && (time(i) > startTime)
        xMeas(end+1) = abs(xCV0(i) - xQTM(i));
        idxXMeas(end+1) = i;
    end
    if (yQTM(i) ~= trans(2)) && (yCV0(i) ~= 0) && (time(i) > startTime)
        yMeas(end+1) = abs(yCV0(i) - yQTM(i));
        idxYMeas(end+1) = i;
    end
end
end

% Extract and display statistics - [mm]
    % Filter
XavgFilter = mean(xFilter)*1000;
XstdFilter = std(xFilter)*1000;
YavgFilter = mean(yFilter)*1000;
YstdFilter = std(yFilter)*1000;
table(XavgFilter, XstdFilter, YavgFilter, YstdFilter)

    % X Measurements
XavgMeas = mean((xMeas))*1000;
XstdMeas = std((xMeas))*1000;
YavgMeas = mean((yMeas))*1000;
YstdMeas = std((yMeas))*1000;
table(XavgMeas, XstdMeas, YavgMeas, YstdMeas)

%% X Translation Plotting

xTransPlot = figure(Name="xTranslationPlot");

% Plot Measured Position from D435 Camera
plot(time,xCV0, '.', 'Color', '#CEB7B7',...
    MarkerSize=mSize)
hold on
% Plot Reference from QTM
plot(t,xQTMi, '.k', MarkerSize=qSize)
% Plot Kalman Filter Estimate Position
plot(t, xKalman, '.r', MarkerSize=kSize)

```

```

[~, icons] = legend('xCV0', 'xQTMinterp', 'xKalman', 'Location','northeast');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

ylabel('Meters')
xlabel('Seconds')
xlim([startTime stopTime])
title('3D Kalman Filter: x Estimate')

set(xTransPlot,'units','normalized','outerposition',[0 0 1 1])
% Export figure
saveas(xTransPlot, ['poseTest_',num2str(testNr), '_xTrans3D'])
saveas(xTransPlot, ['poseTest_',num2str(testNr), '_xTrans3D.png'])

%% Y Translation Plotting
yTransPlot = figure(Name="yTranslationPlot");

% Plot Measured Position from D435 Camera
plot(time,yCV0, '.', 'Color','#C4D383', ...
    MarkerSize=mSize)
hold on
% Plot Reference from QTM
plot(t,yQTMi, '.k', MarkerSize=qSize)
% Plot Kalman Filter Estimate Position
plot(t, yKalman, '.', 'Color', '#1D9300', MarkerSize=kSize)

[~, icons] = legend('yCV0', 'yQTMinterp', 'yKalman', 'Location','northeast');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

ylabel('Meters')
xlabel('Seconds')
xlim([startTime stopTime])
title('3D Kalman Filter: y Estimate')

set(yTransPlot,'units','normalized','outerposition',[0 0 1 1])
% Export figure
saveas(yTransPlot, ['poseTest_',num2str(testNr), '_yTrans3D'])
saveas(yTransPlot, ['poseTest_',num2str(testNr), '_yTrans3D.png'])

%% Z Translation Plotting
zTransPlot = figure(Name="zTranslationPlot");

% Plot Measured Position from D435 Camera
plot(time,zCV0, '.', [109/255, 209/255, 255/255], ...
    MarkerSize=mSize)
hold on
% Plot Reference from QTM
plot(t,zQTMi, '.k', MarkerSize=qSize)
% Plot Kalman Filter Estimate Position
plot(t, zKalman, '.b', MarkerSize=kSize)
[a, icons] = legend('zCV0', 'zQTMinterp', 'zKalman', 'Location','southwest');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)

```

```

ylabel('Meters')
xlabel('Seconds')
xlim([startTime stopTime])
title('3D Kalman Filter: z Estimate')

set(zTransPlot,'units','normalized','outerposition',[0 0 1 1])
% Export figure
saveas(zTransPlot, ['poseTest_',num2str(testNr), '_zTrans3D'])
saveas(zTransPlot, ['poseTest_',num2str(testNr), '_zTrans3D.png'])

%% x Difference Plotting
% Transpose for plotting
xFilter = xFilter';
xMeas = xMeas';

% Plot Differences between filter and no filter
xDiff3D = figure(Name='xDiff3D');
measurements = plot(time(idxXMeas),xMeas,'.', ...
    'Color','#CEB7B7');
hold on
filter = plot(t(idxXf),xFilter,'.r',MarkerSize=1);
title('x Comparison: abs(error) of measurement and filter')

% hold on
diffSize = 1.5;
diffColor = '#7F0000';
measColor = '#FF9191';
yline((XavgMeas+XstdMeas)/1000,'-', 'Color',measColor,'LineWidth',diffSize)
yline((XavgMeas-XstdMeas)/1000,'-', 'Color',measColor,'LineWidth',diffSize)
yline((XavgFilter+XstdFilter)/1000,'--', 'Color', ...
    diffColor,'LineWidth',diffSize)
yline((XavgFilter-XstdFilter)/1000,'--', 'Color', ...
    diffColor,'LineWidth',diffSize)

% Plot NaNs for legend
measLine = plot(nan, nan, '--', 'Color', measColor);
diffLine = plot(nan, nan, '--', 'Color', diffColor);

ylabel('Meters')
xlabel('Seconds')
[~, icons] = legend([measurements filter measLine diffLine], ...
    'xCV0-xQTM', 'xKalman-xQTMi', 'Measurement Confidence Interval', ...
    'Filter Confidence Interval', 'Location','northeast');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)
xlim([startTime stopTime])

set(xDiff3D,'units','normalized','outerposition',[0 0 1 1])
% Export figure
saveas(xDiff3D, ['poseTest_',num2str(testNr), '_xDiff3D'])
saveas(xDiff3D, ['poseTest_',num2str(testNr), '_xDiff3D.png'])

%% y Difference Plotting
% Transpose for plotting
yFilter = yFilter';
yMeas = yMeas';

```

```

% Plot Differences between filter and no filter
yDiff3D = figure(Name='yDiff3D');
measurements = plot(time(idxYMeas),yMeas,'.', ...
    'Color','#C4D383');
hold on
filter = plot(t(idxYf),yFilter,'.', 'Color','#1D9300',MarkerSize=1);
title('y Comparison: abs(error) of measurement and filter')

% hold on
diffSize = 1.5;
diffColor = '#007C1B';
measColor = '#99FFAA';
yline((YavgMeas+YstdMeas)/1000,'-', 'Color',measColor,'LineWidth',diffSize)
yline((YavgMeas-YstdMeas)/1000,'-', 'Color',measColor,'LineWidth',diffSize)
yline((YavgFilter+YstdFilter)/1000,'--', 'Color', ...
    diffColor,'LineWidth',diffSize)
yline((YavgFilter-YstdFilter)/1000,'--', 'Color', ...
    diffColor,'LineWidth',diffSize)

% Plot NaNs for legend
measLine = plot(nan, nan, '--', 'Color', measColor);
diffLine = plot(nan, nan, '--', 'Color', diffColor);

ylabel('Meters')
xlabel('Seconds')
[~, icons] = legend([measurements filter measLine diffLine], ...
    'yCV0-yQTM', 'yKalman-yQTMi', 'Measurement Confidence Interval', ...
    'Filter Confidence Interval', 'Location','northeast');
% Change size of legend icons
icons = findobj(icons, '-property', 'Marker', '-and', '-not', 'Marker', ...
    'none');
set(icons, 'MarkerSize', 20)
xlim([startTime stopTime])

set(yDiff3D,'units','normalized','outerposition',[0 0 1 1])
% Export figure
saveas(yDiff3D, ['poseTest_',num2str(testNr), '_yDiff3D'])
saveas(yDiff3D, ['poseTest_',num2str(testNr), '_yDiff3D.png'])

```

## G.3 Server

### G.3.1 dronePosVec.proto

```
syntax = "proto3";

package dronePosVec;

message dronePosition{
    dataDevices deviceType = 1;
    repeated uint32 posShape = 2           [packed=true];
    repeated float position = 3          [packed=true];
    repeated uint32 rotShape = 4          [packed=true];
    repeated float rotation = 5          [packed=true]; //rotation matrix ...
        from camera, vector from IMU
    uint64 timestamp_ns = 6; //likely mono, so two timestamps needed ...
        to be useful
    uint32 camIteration = 7; //where in array to append cameraRaw to ...
        cameraIMG
    repeated uint32 cameraRaw = 8      [packed=true];
    repeated float rotation2 = 9      [packed=true];
    float droneBattery = 10;
}

enum dataDevices {
    IMUonly = 0;
    CameraPos = 1;
    KalmanFilter = 2;
    CameraImgRGB = 3; //likely unused as other sending method may be ...
        used instead
    CameraImgDepth = 4;
}

message droneControl{ //this goes from -1 to 1
    float motorFL = 1;
    float motorFR = 2;
    float motorBL = 3;
    float motorBR = 4;
    bool killswitch = 5;
}

message dataTransfers{
    progName ID = 1;
    transferType type = 2;
    string msg = 3;
    int64 timeSync_ns = 4;
    string IP = 5; //for python and matlab
    uint32 port = 6;
    bytes sockaddr = 7; //for c++
    uint32 sockaddrlen = 8;
    uint32 sa_family = 9;
}

enum transferType {
    timeSync = 0;
    socketInfo = 1;
    stateChange = 2;
    start = 3;
    end = 4;
}
```

```
enum progName{
    server = 0;
    drone = 1;
    estimator = 2;
    arena = 3;
    camera = 4;
    rl = 5;
}
```

### G.3.2 Server Cmake

CMakeLists.txt for server program

```
cmake_minimum_required(VERSION 3.8)
project(arenaServer)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
    add_compile_options(-Wall -Wextra -Wpedantic)
endif()

#uncomment for debugging:
#set(CMAKE_BUILD_TYPE DEBUG)
#set(CMAKE_CXX_FLAGS_DEBUG "${CMAKE_CXX_FLAGS_DEBUG} -g")
#set(CMAKE_CXX_FLAGS "-g")

find_package(Protobuf 3.12.4 REQUIRED)
include_directories(${Protobuf_INCLUDE_DIRS})
include_directories(${HOME}/.local/include)

add_executable(test src/test.cpp src/dronePosVec.pb.cc)
add_executable(arenaServer src/messengerClass.cpp src/msgThreads.cpp ...
    src/serverMain.cpp src/dronePosVec.pb.cc)

target_link_libraries(test protobuf::libprotobuf protobuf::libprotobuf-lite)
target_link_libraries(arenaServer protobuf::libprotobuf ...
    protobuf::libprotobuf-lite)

#target_include_directories(test PRIVATE ${HOME}/.local/include)
#target_link_libraries(test PRIVATE ${HOME}/.local/lib/libprotobuf.so)

install(TARGETS
    arenaServer
    test
    DESTINATION lib/${PROJECT_NAME})
```

### G.3.3 arenaServer.h

```
#ifndef SOCKETCLASS_H
#define SOCKETCLASS_H

#include <chrono>
#include "dronePosVec.pb.h"
//#include "programSpecifics.h"
#include <sys/socket.h>
#include <string.h>
#include <netdb.h>
#include <thread>
#include <queue>
#include <atomic>
#include <condition_variable>

using ns_t = std::chrono::nanoseconds;

//forward declarations
std::string getSelfIP(const char nameserver[]);
class EstimatorMessenger;
class CameraMessenger;
class DroneMessenger;
class AbMessenger;

enum threadStartType
{
    recvOnly,
    sendOnly,
    sendRecv
};

class ClientSocket {
public:
    virtual ~ClientSocket() = default;
    virtual socklen_t getClient(struct sockaddr* clientAddr) = 0; // ...
    ideally should add all these into one init function
    void initSend();
    virtual int sendDrone(const char* msg, size_t msglen) = 0; //might rename
    virtual ssize_t recvDrone(char* buffer, size_t bufferLen) = 0;
    virtual void setTimeout() = 0;
    virtual void setTimeout(const long int sec, const long int microSec) = 0;
    int dServerConnect();
    int sSyncTimer();
};

class SocketMethods{
public:
    SocketMethods(ns_t timer, std::string addr)
        :serverTimer_(timer)
        ,addr_(addr)
    {
        clientAddrLen_ = sizeof(clientAddr_);
    }
    int getIPfromName(std::string hostname, int port);
    virtual ssize_t initRecv() = 0;
    void genAddrProtoc(dronePosVec::dataTransfers &data);
    int clientConnect(struct sockaddr* clientAddr, socklen_t addrLen);
    ssize_t clientSend(const char* msg, size_t msgSize);
    ssize_t clientRecv(char* buffer, size_t bufferSize);
    void setTimeout();
};
```

```

    void setTimeout(const long int sec,const long int microSec);

protected:
    const ns_t serverTimer_; //needs to be set in constructor
    void sleepToInterval_(ns_t interval); //might be separated into two ...
        functions with get nanoseconds to interval, its nice to have sometimes
    ns_t monoTimeNow_();

    dronePosVec::dataTransfers data_;
    std::string addr_; //likely 127.0.0.1 for all implementations
    int socketSetup_(int port); //creates and binds socket, returns 0 if ...
        successful, must be called in constructor
    int f_socket_;
    struct addrinfo* plocalAddr_ ; //this is needed because getaddrinfo() ...
        wants an addrinfo** type
    //struct addrinfo localAddr_; //breaks everything dont use
    socklen_t localAddrLen_;
    struct sockaddr clientAddr_;
    socklen_t clientAddrLen_;
}; //SocketMethods

class ServerSocket : public SocketMethods {
public:
    ServerSocket(ns_t timer,std::string addr, const int port)
    :SocketMethods(timer,addr)
    ,port_(port)
    {
        //no code
    }
    virtual ~ServerSocket() = default;
    int syncTimer();

    int sendSocketInfo();
    virtual ssize_t initRecv() override;
    ssize_t serverRecvfrom(char* buffer, size_t bufferSize);
    virtual dronePosVec::progName checklistLoop() = 0;
    virtual dronePosVec::progName mainRecvloop() = 0;

    ns_t calcSleepTime(ns_t interval);
    int socketShutdown();

    ns_t getServerTimer();
    void setSocketList(std::vector<std::unique_ptr<AbMessenger>>* ...
        socketClasses);

protected:
    dronePosVec::progName clientProgram_;
    //dronePosVec::transferType clientProgram_; //checklist might be based ...
        on which ID in enum
    const size_t bufferSize_ = 1024;
    char buffer_[1024];
    const int port_;
    std::vector<std::unique_ptr<AbMessenger>>* vpSocketClasses_;

}; //ServerSocket

class AbMessenger : public SocketMethods {
public:
    AbMessenger(ns_t timer, std::string addr, long int recvInterval, long ...
        int sendInterval,std::queue<std::string>* sharedQueue,const ...

```

```

        dronePosVec::progName progType, threadStartType ...
        threadfuncs, std::string stringName)
:SocketMethods(timer, addr)
,strName(stringName)
,recvInterval_(ns_t(recvInterval))
,sendInterval_(ns_t(sendInterval))
,pq1(sharedQueue)
,pq2(sharedQueue)
,classProgram_(progType)
,threadFuncs_(threadfuncs)
{
    //no code
}
AbMessenger(ns_t timer, std::string addr, long int recvInterval, long ...
    int sendInterval, std::queue<std::string>* ...
    sharedQueue1, std::queue<std::string>* sharedQueue2, const ...
    dronePosVec::progName progType, threadStartType ...
    threadfuncs, std::string stringName)
:SocketMethods(timer, addr)
,strName(stringName)
,recvInterval_(ns_t(recvInterval))
,sendInterval_(ns_t(sendInterval))
,pq1(sharedQueue1)
,pq2(sharedQueue2)
,classProgram_(progType)
,threadFuncs_(threadfuncs)
{
    //no code
}
virtual ~AbMessenger() = default;
virtual ssize_t initRecv() override;
//functions to be overrided:
virtual void recvThread(); //generic: recv and add to q
virtual void sendThread(); //generic: send from q2

int startThread(threadStartType startType);
int joinThread();
ssize_t getThreadList(std::thread* tbuffer[], size_t tbufferlen); ...
    //currently unused
dronePosVec::progName getName();
bool getConnection();
void setConnection(bool status);
const std::string strName;
threadStartType getThreadStart();
/*starts this programs threads and also sends a message to the client ...
   to start*/
void conditionNotify(bool startProg);

protected:
    int waitforProgStart_();
    void appendToQueue_(std::queue<std::string>& queueNum, const ...
        std::string& msg);
    int blockingGetQueue_(std::queue<std::string>& queueNum, std::string& ...
        msg, int timeout);
    ns_t recvInterval_; //not const because might change during runtime
    ns_t sendInterval_;
    std::queue<std::string>* const pq1;
    std::queue<std::string>* const pq2;
    volatile std::atomic_bool threadloop_ = true;

    bool connected_ = false;

```

```

const dronePosVec::progName classProgram_;

const threadStartType threadFuncs_;
const size_t bufferSize_ = 1024;
char recvMsg_[1024]; //fix to bufferSize_
char sendMsg_[1024];
std::thread tRecv_;
std::thread tSend_;
std::mutex m_;
std::condition_variable cv_;
}; //AbMessenger

class ServerMain : public ServerSocket {
public:
    ServerMain(ns_t time, std::string addr, const int port)
        : ServerSocket(time, addr, port)
    {
        socketSetup_(port);
    }
    virtual dronePosVec::progName checklistLoop() override;
    virtual dronePosVec::progName mainRecvloop() override;
    struct sockaddr* getClientAddr();
    socklen_t getClientAddrSize();
};

class CameraMessenger : public AbMessenger {
public:
    CameraMessenger(ns_t timer, std::string addr, long int recvInterval, ...
                    long int sendInterval, std::queue<std::string>* sharedQueue)
        : AbMessenger(timer, addr, recvInterval, sendInterval, ...
                      sharedQueue, dronePosVec::camera, ...
                      threadStartType::recvOnly, std::string("Camera"))
    {
        socketSetup_(0);
    }
    ~CameraMessenger()
    {

    }
    //virtual void recvThread() override;
    //virtual void sendThread() override; //unused for camera
}; //CameraMessenger

class EstimatorMessenger : public AbMessenger {
public:
    EstimatorMessenger(ns_t timer, std::string addr, long int ...
                       recvInterval, long int sendInterval, std::queue<std::string>* ...
                       sharedQueue1, std::queue<std::string>* sharedQueue2)
        : AbMessenger(timer, addr, recvInterval, sendInterval, sharedQueue1, ...
                      sharedQueue2, dronePosVec::estimator, ...
                      threadStartType::sendRecv, std::string("Estimator"))
    {
        socketSetup_(0);
    }
}; //EstimatorMessenger

class DroneMessenger : public AbMessenger {
public:
    DroneMessenger(ns_t timer, std::string addr, long int recvInterval, ...
                  long int sendInterval, std::queue<std::string>* ...
                  sharedQueue1, std::queue<std::string>* sharedQueue2)

```

```

:AbMessenger(timer,addr,recvInterval,sendInterval,sharedQueue1, ...
    sharedQueue2,dronePosVec::drone, ...
    threadStartType::sendRecv,std::string("Drone"))
{
    socketSetup_(0);
}
}; //DroneMessenger

class RLMessenger : public AbMessenger {
public:
    RLMessenger(ns_t timer, std::string addr, long int recvInterval, long ...
        int sendInterval,std::queue<std::string>* ...
        sharedQueue1,std::queue<std::string>* sharedQueue2)
:AbMessenger(timer,addr,recvInterval,sendInterval,sharedQueue1, ...
    sharedQueue2,dronePosVec::rl, ...
    threadStartType::sendRecv,std::string("RL"))
{
    socketSetup_(0);
}
}; //DroneMessenger

class ArenaMessenger : public AbMessenger {
public:
    ArenaMessenger(ns_t timer, std::string addr, long int recvInterval, ...
        long int sendInterval,std::queue<std::string>* ...
        sharedQueue1,std::queue<std::string>* sharedQueue2)
:AbMessenger(timer,addr,recvInterval,sendInterval,sharedQueue1, ...
    sharedQueue2,dronePosVec::arena, ...
    threadStartType::sendRecv,std::string("Arena"))
{
    socketSetup_(0);
}
}; //DroneMessenger

#endif //SOCKETCLASS_H

```

#### G.3.4 messengerClass.cpp

Implementation for arenaServer.h

```
#include "dronePosVec.pb.h"
#include "arenaServer.h"
//#include "programSpecifics.h"
#include <sys/socket.h>
#include <chrono>
#include <netdb.h> //for getaddrinfo
#include <arpa/inet.h> //for inet_ntop
//rename this file
#include <cerrno>
#include <condition_variable> //for blocking queues

//std::cout<<"errno:"<<strerror(errno)<<", "<<errno<<std::endl;

std::string getSelfIP(const char nameserver[])
{
    struct addrinfo hints;
    struct addrinfo* ret;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_protocol = IPPROTO_UDP;
    hints.ai_protocol = IPPROTO_UDP;
    hints.ai_flags = AI_CANONNAME;
    //const char uiasearch[] = ".uia.no";
    const size_t hostnamesize = 64;
    char hostName[hostnamesize];
    gethostname(hostName, hostnamesize);
    for (size_t i = 0; i < hostnamesize; i++) //append ".uia.no"
    {
        if (hostName[i] == '\0')
        {
            if ((hostnamesize - i) < sizeof(&nameserver))
            {
                std::cout<<"hostname too long"<<std::endl;
                exit(-1);
            }
            else
            {
                memcpy(&hostName[i], nameserver, sizeof(&nameserver));
                break;
            }
        }
    }
    int r = getaddrinfo(hostName, NULL, &hints, &ret);
    if (r < 0)
    {
        std::cout<<"error when getting hostname or ip, check internet ...
connection"<<std::endl;
        std::cout<<"errno:"<<strerror(errno)<<", "<<errno<<std::endl;
        exit(r);
    }
    //convert to human readable IP
    char strIP[INET_ADDRSTRLEN];

    inet_ntop(AF_INET, &((struct ...
        sockaddr_in*)ret->ai_addr)->sin_addr), &strIP[0], INET_ADDRSTRLEN);
```

```

        std::cout<<"hostname: "<<std::string(ret->ai_canonname)<<", ip: ...
                  "<<std::string(strIP)<<std::endl;
        freeaddrinfo(&hints);
        freeaddrinfo(ret);
        return std::string(strIP);
    }

/* Generic socket methods*/
int SocketMethods::socketSetup_(int port)
{
    struct addrinfo hints;
    memset(&hints,0,sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_protocol = IPPROTO_UDP;
    int r = -1;
    if (port == 0)
    {
        r = getaddrinfo(addr_.c_str(), 0, &hints, &plocalAddr_); //port as ...
                    0 gets automatic unused port
    }
    else
    {
        char decimal_port[16];
        snprintf(decimal_port, 16, "%d", port);
        decimal_port[15] = '\0';
        r = getaddrinfo(addr_.c_str(), decimal_port, &hints, ...
                        &plocalAddr_); //connect with port
    }
    clientAddrLen_ = plocalAddr_->ai_addrlen;
    if(r != 0 || plocalAddr_ == NULL)
    {
        std::cout<<"invalid address or port"<<std::endl;
        return r;
    }
    f_socket_ = socket(plocalAddr_->ai_family, SOCK_DGRAM | SOCK_CLOEXEC, ...
                       IPPROTO_UDP);
    if(f_socket_ == 1)
    {
        std::cout<<"failed to create udp socket"<<std::endl;
    }
    int reusenum = 1;
    setsockopt(f_socket_, SOL_SOCKET, SO_REUSEADDR,&reusenum,1); //allow ...
                    reuse of local address, primarily for main server class
    r = bind(f_socket_, plocalAddr_->ai_addr,plocalAddr_->ai_addrlen);
    if(r != 0)
    {
        std::cout<<"failed to bind socket"<<std::endl;
    }
    localAddrLen_ = sizeof(plocalAddr_);
    return r;
}

int SocketMethods::getIPfromName(std::string hostname,int port) ...
//alternative method for future connectsocket for the future when not ...
using static ip
{
    std::cout<<"getting ip of: "<<hostname<<std::endl;
    struct addrinfo hints;
    memset(&hints,0,sizeof(hints));
    hints.ai_family = AF_INET;

```

```

hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = IPPROTO_UDP;
char decimal_port[16];
snprintf(decimal_port, 16, "%d", port);
decimal_port[15] = '\0';
int r = getaddrinfo(hostname.c_str(), decimal_port, &hints, &plocalAddr_);
if(r != 0 || plocalAddr_ == NULL)
{
    std::cout<<"invalid address or port"<<std::endl;
    return -1;
}
//convert to human readable IP
char strIP[INET_ADDRSTRLEN];

inet_ntop(AF_INET,&(((struct ...
    sockaddr_in*)plocalAddr_->ai_addr)->sin_addr),&strIP[0], ...
    INET_ADDRSTRLEN);
uint32_t port2 = ntohs(((struct ...
    sockaddr_in*)plocalAddr_->ai_addr)->sin_port);
std::cout<<"ip address server: "<<strIP<<":"<<port2<<std::endl;
addr_ = strIP;
return r;
}

void SocketMethods::genAddrProtoc(dronePosVec::dataTransfers &data)
{
    char strIP[INET_ADDRSTRLEN];
    getsockname(f_socket_,plocalAddr_->ai_addr,&plocalAddr_->ai_addrlen);

    inet_ntop(AF_INET,&(((struct ...
        sockaddr_in*)plocalAddr_->ai_addr)->sin_addr),&strIP[0], ...
        INET_ADDRSTRLEN);
    uint32_t port = ntohs(((struct ...
        sockaddr_in*)plocalAddr_->ai_addr)->sin_port);
    std::cout<<"ip address base: "<<strIP<<":"<<port<<std::endl;

    data.Clear();
    data.set_sa_family(plocalAddr_->ai_addr->sa_family); //for sending to ...
        c++ programs
    data.set_sockaddr(plocalAddr_->ai_addr->sa_data);
    data.set_sockaddrlen(plocalAddr_->ai_addrlen);

    data.set_ip(strIP); //for sending to python and matlab programs
    data.set_port(port);//ntohs() converts from network byte order, ...
        (sockaddr_in*)... converts sockaddr to human readable form
    data.set_id(dronePosVec::server);
}

void SocketMethods::sleeptoInterval_(std::chrono::nanoseconds interval)
{
    std::this_thread::sleep_for(interval - ((monoTimeNow_() - serverTimer_) ...
        % interval.count()));
}

std::chrono::nanoseconds SocketMethods::monoTimeNow_()
{
    return std::chrono::duration_cast<std::chrono::nanoseconds>( ...
        std::chrono::steady_clock::now().time_since_epoch());
}

ssize_t SocketMethods::clientSend(const char* msg, size_t msgSize)

```

```

{
    return send(f_socket_,msg,msgSize,0);
}

ssize_t SocketMethods::clientRecv(char* buffer, size_t bufferSize)
{
    ssize_t recvLen = recv(f_socket_, buffer, bufferSize,0);
    buffer[recvLen] = '\0';
    return recvLen;
}

int SocketMethods::clientConnect(struct sockaddr* clientAddr,socklen_t ...
addrLen)
{
    int r = connect(f_socket_,clientAddr,addrLen);
    if (r == 0) //for connecting in the future, useless for initial loop
    {
        memcpy(&clientAddr_,clientAddr,addrLen);
        clientAddrLen_ = addrLen;
    }
    else
    {
        std::cout<<"connection failure"<<std::endl;
        std::cout<<"errno:"<<strerror(errno)<<", "<<errno<<std::endl;
    }
    return r;
}

void SocketMethods::setTimeout()
{
    struct timeval noTimeout;
    noTimeout.tv_sec = 0;
    noTimeout.tv_usec = 0;
    setsockopt(f_socket_,SOL_SOCKET,SO_RCVTIMEO,&noTimeout,sizeof(noTimeout));
}

void SocketMethods::setTimeout(const long int sec,const long int microSec)
{
    struct timeval timeoutTime;
    timeoutTime.tv_sec = sec;
    timeoutTime.tv_usec = microSec;
    setsockopt(f_socket_,SOL_SOCKET,SO_RCVTIMEO, ...
               &timeoutTime,sizeof(timeoutTime));
}

ns_t ServerSocket::getServerTimer()
{
    return serverTimer_;
}
/* socket related functions for AbMessenger */
/* other */

/* Server functions */
ssize_t ServerSocket::initRecv()
{
    setTimeout(5,0);
    ssize_t msgSize;
    msgSize = serverRecvfrom(&buffer_[0],bufferSize_);
    if (msgSize < 0)
    {
        std::cout<<"timeout"<<std::endl;
    }
}

```

```

        return -1;
    }
    std::cout<<"msg recv"<<std::endl; //somewhat temp
try
{
    data_.ParseFromArray(buffer_, msgSize);
    //if no exception:
    std::cout<<"msg received: '" << data_.msg()<< "' with ID: ...
        " <<data_.id() <<std::endl;
    clientProgram_ = data_.id();
    if (clientProgram_ == dronePosVec::server)
    {
        std::cout<<"ERROR: ID not specified in message"<<std::endl;
        return -2;
    }
}
catch(const std::exception& e)
{
    std::cerr << e.what() << '\n';
    std::cout<<"invalid message"<<std::endl;
    return -3;
}
return msgSize;
}

int ServerSocket::socketShutdown()
{
    shutdown(f_socket_, SHUT_RDWR);
    return close(f_socket_);
}

ssize_t ServerSocket::serverRecvfrom(char* buffer, const size_t bufferSize)
{
    ssize_t r = recvfrom(f_socket_, buffer, ...
        bufferSize, 0, &clientAddr_, &clientAddrLen_);
    return r;
}

ns_t ServerSocket::calcSleepTime(ns_t interval)
{
    return (interval - ((monoTimeNow_() - serverTimer_) % interval.count()));
}

dronePosVec::progName ServerMain::mainRecvloop()
{
    clientProgram_ = dronePosVec::server; //default;
    char tempmsg[6] = {'s','e','r','v','r','\0'}; //TODO: make not temporary
    bool recvLooping = true;
    int r = 0;
    while(recvLooping) //restart recv in case of an invalid message such ...
        as from port scanning, but end if timeout
    {
        r = initRecv(); //wait for first recvTo
        if (r > 0)
        {
            memcpy(buffer_, &tempmsg, 6); //temporary
            if ((clientConnect(&clientAddr_, clientAddrLen_)) >= 0)
            {
                clientSend(tempmsg, 6); //temporary msg
                checklistLoop(); //checklist
            }
        }
    }
}

```

```

        std::cout<<"checklist done"<<std::endl;
        //restart mainloop
        socketShutdown();
        socketSetup_(port_);
        recvLooping = false;
        return clientProgram_;
    }
    else
    {
        std::cout<<"failed to connect"<<std::endl;
    }
}
else if (r == -1)
{
    recvLooping = false;
    break;
}

}

return dronePosVec::server; //default
}

dronePosVec::progName ServerMain::checklistLoop()
{
    setTimeout();
    bool looping = true;
    while(looping)
    {
        setTimeout(5,0);
        clientRecv(buffer_,bufferSize_);
        data_.ParseFromArray(buffer_,bufferSize_);
        switch(data_.type())
        {
            case dronePosVec::timeSync:
            {
                std::cout<<"timesync request"<<std::endl;
                syncTimer();
                break;
            }
            case dronePosVec::socketInfo:
            {
                std::cout<<"request for new socket info"<<std::endl;
                sendSocketInfo();
                break;
            }
            case dronePosVec::stateChange:
            {
                //start threads
                std::cout<<"statechange req"<<std::endl;
                looping = false;
                break;
            }
            default:
            {
                looping = false;
                break;
            }
        }
    }
    return data_.id();
}
}

```

```

int ServerSocket::syncTimer()
{
    long int intervalTime = 100000000; //100ms
    data_.Clear();
    data_.set_id(dronePosVec::server);
    data_.set_msg("GlobalTimer");
    data_.set_type(dronePosVec::timeSync);
    data_.SerializeToArray(buffer_, bufferSize_);
    setTimeout(10, 0);

    ns_t sendTime = calcSleepTime(ns_t(intervalTime)); //send next ...
    relative 100ms
    std::this_thread::sleep_for(sendTime);
    clientSend(buffer_, data_.ByteSizeLong());
    sendTime = monoTimeNow_();

    ssize_t msgLen = clientRecv(buffer_, bufferSize_); //read
    if (msgLen > 0)
    {
        data_.Clear();
        data_.ParseFromArray(buffer_, msgLen);
        std::chrono::nanoseconds recvTime = monoTimeNow_();
        long int timeDiff = (recvTime - sendTime).count() - intervalTime;
        std::cout<<"time difference = "<<timeDiff<<std::endl;
        data_.set_id(dronePosVec::server);
        data_.set_msg("offset");
        data_.set_timesync_ns(timeDiff);
        data_.SerializeToArray(buffer_, bufferSize_);
        clientSend(buffer_, data_.ByteSizeLong()); //adding an offset might ...
        be weird, 0.2-0.7ms delay without it
        return 0;
    }
    return -1; //failed to recv msg
}

int ServerSocket::sendSocketInfo()
{
    for(const std::unique_ptr<AbMessenger>& i: *vpSocketClasses_)
    {
        if (i->getName() == clientProgram_)
        {
            i->genAddrProtoc(data_);
            break;
        }
    }
    data_.SerializeToArray(buffer_, bufferSize_);
    return clientSend(buffer_, data_.ByteSizeLong());
}

void ...
    ServerSocket::setSocketList(std::vector<std::unique_ptr<AbMessenger>>* ...
    socketClasses)
{
    vpSocketClasses_ = socketClasses;
}

sockaddr* ServerMain::getClientAddr()
{
    return &clientAddr_;
}

```

```

socklen_t ServerMain::getClientAddrSize()
{
    return clientAddrLen_;
}

ssize_t AbMessenger::initRecv()
{
    setTimeout();
    ssize_t msgSize;
    msgSize = ...
    recvfrom(f_socket_, recvMsg_, bufferSize_, 0, &clientAddr_, &clientAddrLen_);
    try
    {
        data_.ParseFromArray(recvMsg_, msgSize);
    }
    catch(const std::exception& e)
    {
        std::cerr << e.what() << '\n';
        std::cout<<"invalid message"<<std::endl;
        return -1;
    }
    std::cout<<"msg received: '" << data_.msg()<< "' with ID: ...
    "<<data_.id() <<std::endl;
    return clientConnect(&clientAddr_, sizeof(clientAddr_));
}

dronePosVec::progName AbMessenger::getName()
{
    return classProgram_;
}
bool AbMessenger::getConnection()
{
    return connected_;
}

int AbMessenger::joinThread()
{
    //std::cout<<"joining thread of: "<<strName<<std::endl;
    threadloop_ = false;
    if(tRecv_.joinable())
    {
        tRecv_.join();
    }
    if(tSend_.joinable())
    {
        tSend_.join();
    }
    return 0;
}

int AbMessenger::startThread(threadStartType startType)
{
    if (startType == threadStartType::recvOnly)
    { //lambda expression used for capturing class variables
        tRecv_ = std::thread([this]()
        {
            recvThread();
        });

        tRecv_.detach();
    }
}

```

```

    else if (startType == threadStartType::sendOnly)
    {
        tRecv_ = std::thread([this] ()
        {
            recvThread();
        });

        tSend_.detach();
    }
    else
    {
        tRecv_ = std::thread([this] ()
        {
            recvThread();
        });
        tSend_ = std::thread([this] ()
        {
            sendThread();
        });

        tRecv_.detach();
        tSend_.detach();
    }
    return 0;
}

void AbMessenger::appendToQueue_(std::queue<std::string>& queueNum, const ...
    std::string& msg) //unsure if msg should be a reference or not
{
    //should do more to this
    queueNum.push(msg);
}

int AbMessenger::blockingGetQueue_(std::queue<std::string>& ... queueNum, std::string& msg, int timeout_ms)
{
    //std::unique_lock lock(m_);
    ns_t timeNow = monoTimeNow_();
    ns_t timeouttime = ns_t(std::chrono::milliseconds(timeout_ms));
    while (queueNum.empty())
    {
        //std::cout<<strName<<" queue empty"<<std::endl;
        //cv_.wait_for(lock,std::chrono::milliseconds(timeout_ms));
        //really wanted to do this with a conditional variable like the ...
        //above line but would be hard to tell other AbMessenger child ...
        //classes to unlock it
        if ((monoTimeNow_() - timeNow) > timeouttime) //5 seconds timeout
        {
            return -1;
        }
    }
    msg = queueNum.front();
    return 1;
}

int AbMessenger::waitForProgStart_()
{
    std::cout<<strName<<" waiting for program start signal"<<std::endl;
    std::unique_lock lock(m_);
    cv_.wait(lock);
    return 0;
}

```

```

}

void AbMessenger::conditionNotify(bool startProg)
{
    if (!startProg)
    {
        char EOTmsg[1] = {'\4'};
        clientSend(EOTmsg,1);
        threadloop_ = false;
    }
    else
    {
        dronePosVec::dataTransfers data;
        data.Clear();
        data.set_type(dronePosVec::start);
        data.set_msg("start");
        data.set_timesync_ns(sendInterval_.count());
        data.set_id(classProgram_);

        data.SerializeToArray(sendMsg_,bufferSize_);
        clientSend(sendMsg_,data.ByteSizeLong());
    }
    cv_.notify_all(); //can cause "thundering herd" bottleneck but should ...
                      //only last for a short time before threads start sleeping again
}

threadStartType AbMessenger::getThreadStart()
{
    return threadFuncs_;
}

void AbMessenger::setConnection(bool status)
{
    connected_ = status;
}

```

### G.3.5 serverMain.cpp

```
#include "dronePosVec.pb.h"
#include "arenaServer.h"
#include <chrono>
#include <iostream>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <thread>
#include <queue>
#include <vector> //only used for fancily displaying unconnected clients
#include <arpa/inet.h> //for inet_ntop

int main()
{
    const int queueCount = 10;
    std::queue<std::string> queues[queueCount]; //would prefer to have ...
                                                //char* queue type but cant pass length that way

    int unconnected;
    const std::string localAddr = getSelfIP(".uia.no"); //"128.39.200.239"; ...
                                                       //127.0.0.1 //change to resolve selfhost
    ns_t timenow = ...
    std::chrono::duration_cast<ns_t>(std::chrono::steady_clock::now( ...
        ).time_since_epoch()); //get start of global server timer
    ServerMain serverMain(timenow,localAddr,20002);

    std::vector<std::unique_ptr<AbMessenger>> vpMessengers; //vector of ...
                                                               //classes using polymorphism, xxMessenger inherits AbMessenger ...
                                                               //with no new unique functions or data
    vpMessengers.push_back(std::make_unique<CameraMessenger>(... ...
        serverMain.getServerTimer(),localAddr,10000000, ...
        10000000,&queues[0])); // Camera
    vpMessengers.push_back(std::make_unique<EstimatorMessenger>(... ...
        serverMain.getServerTimer(),localAddr,10000000, ...
        10000000,&queues[0],&queues[2])); //Estimator
    vpMessengers.push_back(std::make_unique<DroneMessenger>(... ...
        serverMain.getServerTimer(),localAddr,10000000, ...
        10000000,&queues[0],&queues[1])); //Drone
    vpMessengers.push_back(std::make_unique<RLMessenger>(... ...
        serverMain.getServerTimer(),localAddr,10000000, ...
        10000000,&queues[1],&queues[0])); //RL

    serverMain.setSocketList(&vpMessengers);
    bool connectloop = true;
    while (connectloop)
    {
        std::cout<<"unconnected clients: ";
        unconnected = 0;
        for(const std::unique_ptr<AbMessenger>& i: vpMessengers) //range ...
                                                               //based for loop
        {
            if (i->getConnection() == false)
            {
                unconnected++;
                std::cout<<i->strName<<", ";
            }
        }
        std::cout<<std::endl;
    }
}
```

```

dronePosVec::progName connectedID = serverMain.mainRecvloop(); ...
//-----main socket connection-----
if (connectedID == dronePosVec::server)
{
    connectloop = false;
    break;
}
std::cout<<"connected ID: " <<connectedID<<std::endl;

for(const std::unique_ptr<AbMessenger>& i: vpMessengers) //range ...
    based for loop
{
    if (i->getName() == connectedID)
    {
        //std::cout<<i->strName<<std::endl;
        if(i->initRecv() >= 0) //specific client connection
        {
            i->setConnection(true);
            i->startThread(i->getThreadStart());
            break;
        }
    }
}
if (unconnected == 0)
{
    connectloop = false; //would rather have a check that allows ...
    clients to be reconnected in case of a connection loss
}
}

std::cout<<"type '0' to end program, '1' to start"<<std::endl;
bool mainlooping = true;
int a;
while(mainlooping)
{

    std::cin>>a;

    switch (a)
    {
    case 1:
        //start
        std::cout<<"starting program"<<std::endl;
        for(const std::unique_ptr<AbMessenger>& i: vpMessengers)
        {
            i->conditionNotify(true);
        }
        break;

    default:
        mainlooping = false;
        //finish:
        std::cout<<"ending threads"<<std::endl;
        for(const std::unique_ptr<AbMessenger>& i: vpMessengers) //end ...
            all threads (kind of slow because it does it one by one... ...
            might take 1000ms)
        {
            i->conditionNotify(false);
            i->joinThread();
        }
        break;
    }
}

```

```
        }
    }
    return 0;
}
```

### G.3.6 msgThreads.cpp

implementation of threads for arenaServer.h

```
#include "dronePosVec.pb.h"
#include "arenaServer.h"
#include <thread>
#include <queue>
#include <cerrno>
/*
GENERIC THREAD RECV FUNCTION, OVERRIDE IF NEEDED
*/
void AbMessenger::recvThread()
{
    std::cout<<strName<<" recv thread up"<<std::endl;

    //dont start until main thread sends signal to start threads
    waitForProgStart_();

    dronePosVec::dronePosition data;
    setTimeout();
    setTimeout(5,0);
    ssize_t msgsize = 0;
    char errorStr[] = {'\0'};
    while(threadloop_)
    {
        data.Clear();
        msgsize = clientRecv(recvMsg_,bufferSize_);
        if (msgsize > 0)
        {
            try
            {
                data.ParseFromArray(recvMsg_,msgsize);
                //std::cout<<strName<<" msg recieived: ...
                //<<std::string(recvMsg_,msgsize)<<"\r"; //TEMP
                appendToQueue_(*pq1,std::string(recvMsg_,msgsize));
            }
            catch(const std::exception& e)
            {
                std::cerr << e.what() << '\n'; //incorrect message, ...
                //possibly caused by port scanning or error on other ...
                //process, thread should continue
                appendToQueue_(*pq1,std::string(errorStr,1)); //append ...
                //empty string
            }
        }
        else
        {
            std::cout<<strName<<" recv fail, errno:"<<strerror(errno)<<": ...
            "<<errno<<std::endl;
            threadloop_ = false;
            break;
        }
        sleepToInterval_(recvInterval_);
    }
    std::cout<<strName<<" recv thread ready to join"<<std::endl;
}

void AbMessenger::sendThread()
{
    std::cout<<strName<<" send Thread up"<<std::endl;
```

```

//dont start until main thread sends signal to start threads
waitForProgStart_();

dronePosVec::dronePosition data;
int r = 0;
std::string msg;
while(threadloop_)
{
    //data.Clear();
    r = blockingGetQueue_(*pq2,msg,5000);
    if (r == -1) //timeout from blocking queue
    {
        std::cout<<strName<<" timeout from queue get,"<< " rval: ...
            "<<r<<std::endl;
        threadloop_ = false;
        break;
    }
    else if (msg[0] == '\0')
    {
        //will happen if any wrong messages get passed through, just ...
        ignore for now
        std::cout<<strName<<" incorrect message recieved from ...
            queue,"<< " rval: " <<r<<std::endl;
        pq2->pop();
    }
    else
    {
        //std::cout<<strName<<" msg sent: "<<msg<<std::endl; //TEMP
        clientSend(msg.c_str(),msg.size());
        pq2->pop();
    }
    sleeptoInterval_(sendInterval_);
}
std::cout<<strName<<" send Thread ready to join"<<std::endl;
}

```

## G.4 Clients - C++

This section contains primarily code for the drone, however the drone code is mostly an example of using the client headers.

### G.4.1 drone cmake

CMakeLists.txt for drone program, within a ./cmake/Modules file is a Findpigpio.cmake file which gives the location of the pigpio library files, this is included in the pigpio library.

```
cmake_minimum_required(VERSION 3.8)
project(droneClient)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
    add_compile_options(-Wall -Wextra -Wpedantic)
endif()

set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR aarch64)

#uncomment for debugging:
#set(CMAKE_BUILD_TYPE DEBUG)
#set(CMAKE_CXX_FLAGS_DEBUG "${CMAKE_CXX_FLAGS_DEBUG} -g")
#set(CMAKE_CXX_FLAGS "-g")

find_package(Protobuf 3.12.4 REQUIRED)

list(APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake/Modules)

include_directories(${Protobuf_INCLUDE_DIRS})
include_directories(${HOME}/.local/include)

add_executable(droneClient src/droneMain.cpp src/clientImplementation.cpp ...
    src/threadFuncs.cpp src/IMUImplementation.cpp src/dronePosVec.pb.cc)
target_link_libraries(droneClient pigpio)

target_link_libraries(droneClient protobuf::libprotobuf ...
    protobuf::libprotobuf-lite)

install(TARGETS
    droneClient
    DESTINATION lib/${PROJECT_NAME})
```

#### G.4.2 arenaHeader.h

General client header.

```
#ifndef ARENAHEADER_H
#define ARENAHEADER_H

#include <chrono>
#include "dronePosVec.pb.h"
#include <sys/socket.h>
#include <string.h>
#include <netdb.h>
#include <thread>
#include <atomic>
#include <queue>
#include <condition_variable> //for blocking queues

using ns_t = std::chrono::nanoseconds;

enum threadStartType
{
    recvOnly,
    sendOnly,
    sendRecv
};

class SocketMethods{
public:
    SocketMethods(std::string addr, dronePosVec::progName clientName)
        :addr_(addr),
        ,clientName_(clientName)
    {
        clientAddrLen_ = sizeof(clientAddr_);
    }

    void genAddrProtoc(dronePosVec::dataTransfers &data);
    int clientConnect(struct sockaddr* clientAddr, socklen_t addrLen);
    ssize_t clientSend(const char* msg, size_t msgSize);
    ssize_t clientRecv(char* buffer, size_t bufferSize);
    ssize_t clientRecv(char* buffer, size_t bufferSize, bool pass);
    void setTimeout();
    void setTimeout(const long int sec, const long int microSec);
    int socketShutdown();

protected:
    ns_t serverTimer_ = ns_t(0);
    ns_t timerOffset_;
    void sleeptoInterval_(ns_t interval); //might be separated into two ...
    functions with get nanoseconds to interval, its nice to have sometimes
    ns_t monoTimeNow_();
    std::string serverAddr_;

    dronePosVec::dataTransfers data_;
    void getIPfromName(std::string hostname, int port);
    std::string addr_; //likely 127.0.0.1 for all implementations
    int socketSetup_(int port); //creates and binds socket, returns 0 if ...
    successful, must be called in constructor
    int f_socket_;
    struct addrinfo* plocalAddr_ ; //this is needed because getaddrinfo() ...
    wants an addrinfo** type
```

```

    struct addrinfo* pServerAddr_ ;
    //struct addrinfo localAddr_ ; //breaks everything dont use
    socklen_t localAddrLen_ ;
    struct sockaddr clientAddr_ ;
    socklen_t clientAddrLen_ ;
    const dronePosVec::progName clientName_ ;
} ; //SocketMethods

class ClientClass : protected SocketMethods
{
public:
    ClientClass(std::string addr,dronePosVec::progName ...
                clientName, std::string serverAddr, int serverPort,threadStartType ...
                threadFuncs)
    :SocketMethods(addr, clientName)
    ,threadFuncs(threadFuncs)
    ,serverPort_(serverPort)
    {
        getIPfromName(serverAddr,serverPort_) ;
        socketSetup_(0) ;
    }
    int connectServer();
    int checkList();

    void recvThread();
    void sendThread();
    int startThread(threadStartType startType);
    int joinThread();

    threadStartType getThreadStart(); //TODO: make
    const threadStartType threadFuncs;
    std::queue<std::string> sendQueue;
    std::atomic_bool readingQueue = false;
    volatile std::atomic_bool threadloop = true; //this needs to be atomic ...
                                                as it may be chagned and read at the same time at program end
    void conditionNotify();

private:
    //int connectNewServer_();

    int blockingGetQueue_(std::queue<std::string>& queueNum, std::string& ...
                           msg, int timeout_ms);
    std::mutex m_;
    std::condition_variable cv_;

    void getNextAddr_();
    int cSyncTime_();
    int waitForStartSignal_(bool passRecv);

    int statechange_();
    struct sockaddr nextAddress_;
    socklen_t nextAddrLen_ = 0;
    const int serverPort_;

    ssize_t initSend(char* msg, size_t msgLen);
    const size_t bufferSize_ = 1024;
    char recvMsg_[1024]; //fix to bufferSize_
    char sendMsg_[1024]; // do these really need to be that big?

    ns_t sendInterval_ = ns_t(100000000); //TODO: temp
    std::thread tRecv_;

```

```
    std::thread tSend_;  
}; //ClientClass  
  
#endif //ARENAHEADER_H
```

### G.4.3 droneMain.cpp

The main file for the drone client

```
#include "arenaHeader.h"
#include "droneIMU.h"
#include "dronePosVec.pb.h"
#include <iostream>

volatile bool mainExit = false;
void sig_handler(int signum);

int main()
{
    int spiHandle = spiInit();
    if (spiHandle < 0)
    {
        std::cout<<"\n pigpio failed to initialize"<<std::endl;
        return -1;
    }

    GyroIMU gyro(spiHandle);
    AccelIMU accel(spiHandle);
    dronePosVec::dronePosition dp;

    //initialize dp
    dp.set_devicetype(dronePosVec::dataDevices::IMUonly);
    dp.add_rotshape(1); //1x3 vector
    dp.add_rotshape(3);
    dp.add_posshape(1); //1x3 vector
    dp.add_posshape(3);

    //ClientClass droneClass("dronearena.uia.no",dronePosVec::drone, ...
    //    "128.39.200.239",20002,threadStartType::sendOnly);
    ClientClass droneClass("10.245.30.158",dronePosVec::drone, ...
        "bl2.uia.no",20002,threadStartType::sendRecv); //temp sendRecv, ...
    //normally sendOnly for IMU
    if (droneClass.connectServer() == 0)
    {
        if (droneClass.checkList() < 0)
        {
            std::cout<<"failure during checklist"<<std::endl;
            return -1;
        }
    }
    else
    {
        std::cout<<"failed to connect"<<std::endl;
        return -1;
    }
    droneClass.startThread(droneClass.threadFuncs);
    signal(SIGINT, &sig_handler);
    while(droneClass.threadloop) //TODO: make some way of exiting safely
    {
        accel.readData();
        gyro.readData();
        accel.populateProtobuf(dp);
        gyro.populateProtobuf(dp); //timestamp is only set here
        if ((droneClass.sendQueue.size() > 0) & (droneClass.readingQueue ...
            == false))
        {
    }
```

```

    try
    {
        droneClass.sendQueue = {};//clear queue
    }
    catch(...)

    {
        //do nothing, .pop will throw an exception rarely when ...
        //thread calls .pop right before this .pop is called but ...
        //not before .size returns 0, not a dangerous error
    }
}

droneClass.sendQueue.push(dp.SerializeAsString());
droneClass.conditionNotify();//tell thread that its safe to read ...
queue (..... this is basically just the same as ...
using an atomic_string at this point)

//std::this_thread::sleep_for(std::chrono::microseconds(50));
if (mainExit)
{
    droneClass.threadloop = false;
}
}

droneClass.joinThread();
gpioEnd(spiHandle);//needs to be called before end of program
return 0;
}

void sig_handler(int signum)
{
    if (signum == SIGINT)
    {
        mainExit = true;
        std::cerr << "Interrupt signal (" << signum << ") received.\n";
    }
    else
    {
        //restore default signal handler
        signal(signum, SIG_DFL);
    }
}

```

#### G.4.4 clientimplementation.cpp

Implementation for the general header class.

```
#include "arenaHeader.h"
#include "dronePosVec.pb.h"
#include <arpa/inet.h> //for inet_ntop
#include <thread>
#include <cerrno>
#include <queue>
#include <condition_variable>

int SocketMethods::socketSetup_(int port) //TODO: add freeaddrinfo()
{
    struct addrinfo hints;
    memset(pServerAddr_, 0, sizeof(&pServerAddr_));
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_protocol = IPPROTO_UDP;
    hints.ai_flags = AI_CANONNAME; //retrieve the canonical name //unused, ...
    usage was moved elsewhere
    int r = -1;
    if (port == 0)
    {
        r = getaddrinfo(addr_.c_str(), 0, &hints, &plocalAddr_); //port as ...
        0 gets automatic unused port
    }
    else
    {
        char decimal_port[16];
        snprintf(decimal_port, 16, "%d", port); //convert to format that ...
        getaddrinfo() wants
        decimal_port[15] = '\0';
        r = getaddrinfo(addr_.c_str(), decimal_port, &hints, ...
            &plocalAddr_); //generate addrinfo with port
    }
    if(r != 0 || plocalAddr_ == NULL)
    {
        std::cout<<"invalid address or port"<<std::endl;
        std::cout<<"errno = "<<strerror(errno)<<" errno val: "<< errno<<", ...
        rVal = "<<r<<std::endl;
        exit(r);
    }
    //exit(r);//temp
    //convert to human readable IP:
    char strIP[INET_ADDRSTRLEN];

    inet_ntop(AF_INET,&((struct ...
        sockaddr_in*)plocalAddr_->ai_addr)->sin_addr),&strIP[0], ...
        INET_ADDRSTRLEN);
    std::cout<<"self addr: " <<strIP<<std::endl;
    addr_ = std::string(strIP); //update hostname to ip addr

    f_socket_ = socket(plocalAddr_->ai_family, SOCK_DGRAM | SOCK_CLOEXEC, ...
        IPPROTO_UDP);
    if(f_socket_ < 0)
    {
        std::cout<<"failed to create udp socket"<<std::endl;
        exit(r);
    }
}
```

```

int reusenum = 1;
r = setsockopt(f_socket_, SOL_SOCKET, SO_REUSEADDR, ...
    &reusenum, sizeof(reusenum)); //allow reuse of local address, ...
    primarily for main server class
if(r != 0)
{
    std::cout<<"failed to set sock option"<<std::endl;
    exit(r);
}
r = bind(f_socket_, plocalAddr_->ai_addr,plocalAddr_->ai_addrlen);
if(r != 0)
{
    std::cout<<"failed to bind socket"<<std::endl;
    exit(r);
}
localAddrLen_ = sizeof(plocalAddr_);
//freeaddrinfo(&hints);
return r;
}

void SocketMethods::getIPfromName(std::string hostname,int port)
{
//std::cout<<"getting ip of: "<<hostname<<std::endl;
struct addrinfo hints;
memset(&hints,0,sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = IPPROTO_UDP;
char decimal_port[16];
snprintf(decimal_port, 16, "%d", port);
decimal_port[15] = '\0';
int r = getaddrinfo(hostname.c_str(), decimal_port, &hints, ...
    &pServerAddr_);
if(r != 0 || pServerAddr_ == NULL)
{
    std::cout<<"invalid address or port"<<std::endl;
    exit(r);
}
//convert to human readable IP to strIP
char strIP[INET_ADDRSTRLEN];
inet_ntop(AF_INET,&((struct ...
    sockaddr_in*)pServerAddr_->ai_addr)->sin_addr),&strIP[0], ...
    INET_ADDRSTRLEN);
uint32_t port2 = ntohs(((struct ...
    sockaddr_in*)pServerAddr_->ai_addr)->sin_port);
std::cout<<"address server: " <<strIP<<" :"<<port2<<std::endl;
serverAddr_ = std::string(strIP);
//freeaddrinfo(&hints);
}

void SocketMethods::genAddrProtoc(dronePosVec::dataTransfers &data)
{
char strIP[INET_ADDRSTRLEN];
getsockname(f_socket_,plocalAddr_->ai_addr,&plocalAddr_->ai_addrlen);

inet_ntop(AF_INET,&((struct sockaddr_in) ...
    plocalAddr_->ai_addr)->sin_addr),&strIP[0],INET_ADDRSTRLEN);
uint32_t port = ntohs(((struct ...
    sockaddr_in*)plocalAddr_->ai_addr)->sin_port);
std::cout<<"ip address base: " <<strIP<<" :"<<port<<std::endl;
}

```

```

        data.Clear();
        data.set_sa_family(((struct sockaddr_in*)plocalAddr_)->sin_family);
        data.set_sockaddr(plocalAddr_->ai_addr->sa_data); //for sending to c++ ...
            programs
        data.set_sockaddrlen(plocalAddr_->ai_addrlen);
        data.set_ip(strIP); //for sending to python and matlab programs
        data.set_port(port); //ntohs() converts from network byte order, ...
            (sockaddr_in*)... converts sockaddr to human readable form
        data.set_id(clientName_);
    }

void SocketMethods::sleeptoInterval_(ns_t interval)
{
    std::this_thread::sleep_for(interval - ((monoTimeNow_() - serverTimer_) ...
        % interval.count()));
}

ns_t SocketMethods::monoTimeNow_()
{
    return std::chrono::duration_cast<ns_t> ...
        (std::chrono::steady_clock::now().time_since_epoch());
}

ssize_t SocketMethods::clientSend(const char* msg, size_t msgSize)
{
    return send(f_socket_,msg,msgSize,0);
}

ssize_t SocketMethods::clientRecv(char* buffer, size_t bufferSize)
{
    ssize_t recvLen = recv(f_socket_, buffer, bufferSize,0);
    buffer[recvLen] = '\0';
    return recvLen;
}

ssize_t SocketMethods::clientRecv(char* buffer, size_t bufferSize,bool pass)
{
    ssize_t recvLen;
    if (pass == true)
    {
        recvLen = recv(f_socket_, buffer, bufferSize,MSG_PEEK);
    }
    else
    {
        recvLen = clientRecv(buffer, bufferSize);
    }

    buffer[recvLen] = '\0';
    return recvLen;
}

int SocketMethods::clientConnect(struct sockaddr* clientAddr,socklen_t ...
    addrLen)
{
    int r = connect(f_socket_,clientAddr,addrLen);
    if (r == 0) //for connecting in the future, useless for initial loop
    {
        memcpy(&clientAddr_,clientAddr,addrLen);
        clientAddrLen_ = addrLen;
    }
    else
}

```

```

    {
        std::cout<<"connection failure"<<std::endl;
        std::cout<<"errno:"<<strerror(errno)<<", "<<errno<<std::endl;
    }
    return r;
}

void SocketMethods::setTimeout ()
{
    struct timeval noTimeout;
    noTimeout.tv_sec = 0;
    noTimeout.tv_usec = 0;
    setsockopt(f_socket_, SOL_SOCKET, SO_RCVTIMEO, ...
                &noTimeout, sizeof(noTimeout));
}

void SocketMethods::setTimeout (const long int sec, const long int microSec)
{
    struct timeval timeoutTime;
    timeoutTime.tv_sec = sec;
    timeoutTime.tv_usec = microSec;
    setsockopt(f_socket_, SOL_SOCKET, SO_RCVTIMEO, ...
                &timeoutTime, sizeof(timeoutTime));
}

int SocketMethods::socketShutdown ()
{
    shutdown(f_socket_, SHUT_RDWR);
    return close(f_socket_);
}
/*
-----CLIENTCLASS-----
*/
int ClientClass::connectServer ()
{
    struct sockaddr serverAddress;
    socklen_t serverAddrLen = sizeof(serverAddress);

    data_.Clear();
    data_.set_id(clientName_);
    data_.set_msg("drone hi");
    data_.SerializeToArray(sendMsg_, bufferSize_);
    //std::cout<<data_.SerializeAsString()<<std::endl;

    setTimeout(3,0);
    int r = 0;
    for(int i = 0; i < 10; i++)
    {
        r = initSend(sendMsg_, data_.ByteSizeLong());
        if (r<0)
        {
            std::cout<<"failed to send initial message"<<std::endl;
            std::cout<<" errno = "<<strerror(errno)<<". errno val: "<< ...
            errno<<", rVal = "<<r<<std::endl;
        }
        r = recvfrom(f_socket_, recvMsg_, bufferSize_, 0, ...
                    &serverAddress,&serverAddrLen);
        if (r > 0)
        {
            std::cout<<"msg recv: "<<recvMsg_<<std::endl;
            break;
        }
    }
}

```

```

        }
    else
    {
        std::cout<<"timeout, retrying: " <<i<<, errno = ...
            "<<strerror(errno)<<", rVal = "<<r<<std::endl;
    }
}
if(r < 0)
{
    return -1;
}
else
{
    int connstat = clientConnect(&serverAddress,serverAddrLen);
    return connstat;
}
}

ssize_t ClientClass::initSend(char* msg, size_t msgLen)
{
    return sendto(f_socket_,msg,msgLen,0, ...
        pServerAddr_->ai_addr,pServerAddr_->ai_addrlen);
}

int ClientClass::checkList() //checklist before starting streaming
{
    std::cout<<"connection sucess"<<std::endl;
    bool checklistLoop = true;
    while(checklistLoop)
    {
        if (serverTimer_.count () == 0)
        {
            std::cout<<"timer missing, getting"<<std::endl;
            data_.Clear();
            data_.set_id(clientName_);
            data_.set_type(dronePosVec::timeSync);
            data_.set_msg("syncReq");
            data_.SerializeToArray(sendMsg_,bufferSize_);
            clientSend(sendMsg_,data_.ByteSizeLong());
            cSyncTime_();
        }
        if (nextAddrLen_ == 0)
        {
            std::cout<<"next address missing, getting"<<std::endl;
            getNextAddr_();
        }
        else
        {
            std::cout<<"checklist complete, requesting state ...
                change"<<std::endl;
            if (statechange_() > 0)
            {
                std::cout<<"connection with new socket success"<<std::endl;
            }
            checklistLoop = false;
            return 0;
        }
    }
    return -1;
//droneClient.mainloop(&dataMsg);
}

```

```

int ClientClass::cSyncTime_()
{
    ns_t intervalTime = ns_t(1000000000);

    setTimeout();
    data_.Clear();
    ssize_t msgLen = clientRecv(recvMsg_, bufferSize_);
    data_.ParseFromArray(recvMsg_, msgLen);
    serverTimer_ = monoTimeNow_();

    data_.set_id(clientName_);
    data_.SerializeToArray(sendMsg_, bufferSize_);
    size_t byteSize = data_.ByteSizeLong();

    sleepToInterval_(intervalTime);
    clientSend(sendMsg_, byteSize);

    msgLen = clientRecv(recvMsg_, bufferSize_);
    data_.ParseFromArray(recvMsg_, msgLen);
    timerOffset_ = std::chrono::nanoseconds(data_.timesync_ns());

    serverTimer_ = serverTimer_ - timerOffset_;
    //std::cout<<"offset: "<< timerOffset_.count ()<<std::endl;
    return 0;
}

void ClientClass::getNextAddr_() //TODO: CONTINUE AND FIX HERE
{
    data_.Clear();
    data_.set_id(clientName_);
    data_.set_type(dronePosVec::socketInfo);
    data_.set_msg("socketReq");
    data_.SerializeToArray(sendMsg_, bufferSize_);
    clientSend(sendMsg_, data_.ByteSizeLong());

    setTimeout();
    int msgLen = clientRecv(recvMsg_, bufferSize_);
    data_.Clear();
    data_.ParseFromArray(recvMsg_, msgLen);
    std::cout<<"next address: "<<data_.ip()<<" :"<<data_.port ()<<std::endl;

    memcpy(&nextAddress_.sa_data,data_.sockaddr().c_str(),data_.sockaddrlen());
    nextAddress_.sa_family = data_.sa_family();
    nextAddrLen_ = data_.sockaddrlen();
}

int ClientClass::statechange_() //TODO: fix
{
    data_.Clear();
    data_.set_id(clientName_);
    data_.set_type(dronePosVec::stateChange);
    data_.set_msg("stateChangeReq");
    data_.SerializeToArray(sendMsg_, bufferSize_);
    clientSend(sendMsg_, data_.ByteSizeLong());

    setTimeout();
    //int msgLen = clientRecv(recvMsg_, bufferSize_);
    socketShutdown();
    //generate new socket and connect to next socket
    int r = socketSetup_(0);
}

```

```

    if (r != 0)
    {
        std::cout<<"failure to set up socket"<<std::endl;
        return -1;
    }
    clientConnect (&nextAddress_,nextAddrLen_);

    data_.Clear();
    data_.set_id(clientName_);
    data_.set_msg("hi");
    data_.SerializeToArray(sendMsg_,bufferSize_);
    return clientSend(sendMsg_,data_.ByteSizeLong());
}

int ClientClass::startThread(threadStartType startType)
{
    std::cout<<"starting threads"<<std::endl;
    if (startType == threadStartType::recvOnly)
    {
        tRecv_ = std::thread([this] ()
        {
            recvThread();
        });

        //std::thread(&AbMessenger::recvThread, this);
        tRecv_.detach();
    }
    else if (startType == threadStartType::sendOnly)
    {
        tSend_ = std::thread([this] ()
        {
            sendThread();
        });

        tSend_.detach();
    }
    else
    {
        tRecv_ = std::thread([this] ()
        {
            recvThread();
        });
        tSend_ = std::thread([this] ()
        {
            sendThread();
        });

        tRecv_.detach();
        tSend_.detach();
    }
    return 0;
}

//threads should be guaranteed to end soon after threadloop is called
int ClientClass::joinThread()
{
    std::cout<<"joining threads"<<std::endl;
    threadloop = false;
    bool waiting = true;
    while(waiting)
    {

```

```

        if (tRecv_.joinable())
        {
            tRecv_.join();
        }
        else if (tRecv_.joinable())
        {
            tRecv_.join();
        }
        else
        {
            waiting = false;
        }
    }
    freeaddrinfo(plocalAddr_);
    freeaddrinfo(pServerAddr_);
    return 0;
}

int ClientClass::waitForStartSignal_(bool passRecv)
{
    std::cout<<"waiting for signal to start program"<<std::endl;
    setTimeout(120,0);
    data_.Clear();
    ssize_t msgLen = 0;
    try
    {
        if (passRecv)
        {
            msgLen = clientRecv(recvMsg_,bufferSize_,true);
        }
        else
        {
            msgLen = clientRecv(recvMsg_,bufferSize_);
        }
        if (recvMsg_[0] != '\4')
        {
            data_.ParseFromArray(recvMsg_,msgLen);
        }
        else
        {
            return -1;
        }
    }
    catch(const std::exception& e)
    {
        std::cerr << e.what() << '\n';
        threadloop = false;
        return -1;
    }
    if (data_.type() == dronePosVec::start)
    {
        if (data_.timesync_ns() > 0)
        {
            sendInterval_ = ns_t(data_.timesync_ns()); //TODO: give both
        }
        return 0;
    }
    return -1;
}

int ClientClass::blockingGetQueue_(std::queue<std::string>& ...

```

```
queueNum, std::string& msg, int timeout_ms)
{
    std::unique_lock lock(m_);
    cv_.wait_for(lock, std::chrono::milliseconds(timeout_ms));
    /*
    while (queueNum.empty())
    {
        cv_.wait_for(lock, std::chrono::milliseconds(timeout_ms));
        return -1;
    }
    */
    msg = queueNum.front();
    return 1;
}

void ClientClass::conditionNotify()
{
    if (readingQueue == true)
    {
        cv_.notify_one();
    }
}
```

#### G.4.5 droneIMU.h

Header for the BMI088 IMU related functions.

```
#ifndef DRONEIMU_H
#define DRONEIMU_H

#include "dronePosVec.pb.h"
#include <thread>
#include <chrono>

//pin defs
#define LED 18 //GPIO 18, physical pin 12
#define CS_ACCEL 12 //GPIO 12, physical pin 22
#define CS_GYRO 13
//spi flags:
#define SPI_CHANNEL 1
#define SPI_SPEED 1000000
#define SPI_MODE_0 0x0
#define SPI_WORDLEN_16 (0x10<<16)
#define SPI_DISABLE_UX ((0x1<<5) | (0x1<<6) | (0x1<<7))
#define SPI_AUXILIARY (0x1<<8)

//registers:
#define READ_BIT (1<<7)
#define ACC_PWR_CTRL 0x7D
#define ACC_X_LSB 0x12
#define ACC_RANGE 0x41
#define GYRO_RANGE 0x0F
#define RATE_X_LSB 0x02
#define GYRO_BANDWIDTH 0x10
#define ACC_CONF 0x40

void gpioEnd(int spiHandle);
int spiInit(); //push to class but leave for now

using ns_t = std::chrono::nanoseconds;

class IIMU
{
public:
    IIMU(int handle)
        :spiHandle_(handle)
    {
    }

    //virtual void readData(int spiHandle,char* txBuffer,char* rxBuffer, ...
    //    struct outputVals& vals) = 0 ;
    virtual int writeSingleReg() = 0;
    virtual void populateProtobuf(dronePosVec::dronePosition& dp) = 0;

protected:
    virtual int sensorInit_(int spiHandle, char* buffer) = 0;
    ns_t monoTimeNow_();
    virtual void calib_() = 0;

    const int spiHandle_;
    struct outputVals
    {
        float xf,yf,zf = 0;
        int16_t xi,yi,zi = 0;
    }
};
```

```

        ns_t t;
    } offsetVals_, sensorVals_;
    float range_;
    const size_t bufferSize = 16;
    char txBuffer_[16];
    char rxBuffer_[16];

}; //IIMU

class GyroIMU : protected IIMU
{
public:
    GyroIMU(int handle)
    :IIMU(handle)
    {
        sensorInit_(spiHandle_,txBuffer_);
        calib_();
    }
    void readData();
    virtual int writeSingleReg() override;
    virtual void populateProtobuf(dronePosVec::dronePosition& dp) override;

protected:
    virtual int sensorInit_(int spiHandle, char* buffer) override;
    virtual void calib_() override;

}; //GyroIMU

class AccelIMU : protected IIMU
{
public:
    AccelIMU(int handle)
    :IIMU(handle)
    {
    }
    void readData();
    virtual int writeSingleReg() override;
    virtual void populateProtobuf(dronePosVec::dronePosition& dp) override;

protected:
    virtual int sensorInit_(int spiHandle,char* buffer) override;
    virtual void calib_() override;

}; //AccelIMU

#endif //DRONEIMU_H

```

#### G.4.6 IMUImplementation.cpp

Implementation for the IMU header file. Using auxiliary SPI

```
#include "droneIMU.h"
#include <pigpio.h>
#include <iostream>
#include <cmath>

void gpioEnd(int spiHandle)
{
    spiClose(spiHandle);
    gpioTerminate(); // call when done with library
}

int spiInit()
{
    if(gpioInitialise()<0) // must be initialized first
    {
        std::cout<<"initialisation failed";
        return -1;
    }
    int spiHandle = spiOpen(SPI_CHANNEL,SPI_SPEED,SPI_WORDLEN_16 | ...
                           SPI_AUXILIARY | SPI_DISABLE_UX | SPI_MODE_0); // using mode 0, ...
    disable ux as cs, enable auxilarary spi, make 16 bit word length
//int spiHandle = spiOpen(SPI_CHANNEL,SPI_SPEED,SPI_AUXILIARY | ...
//                           SPI_DISABLE_UX | SPI_MODE_0); // using mode 0, disable ux as cs, ...
//    enable auxilarary spi, make 16 bit word length
    if (spiHandle<0)
    {
        std::cout<<"spi open failed";
        return -2;
    }
    gpioSetMode(LED, PI_OUTPUT);
    gpioSetMode(CS_ACCEL, PI_OUTPUT);
    gpioSetMode(CS_GYRO, PI_OUTPUT);

    gpioWrite(CS_ACCEL,1);
    gpioWrite(CS_GYRO,1);
    gpioWrite(LED,0); //led for any debugging, TODO: should be removed ...
                      later though

    std::this_thread::sleep_for(std::chrono::milliseconds(1)); //wait 1ms ...
    after startup
    return spiHandle;
}

ns_t IIMU::monoTimeNow_()
{
    return std::chrono::duration_cast<ns_t>( ...
        std::chrono::steady_clock::now().time_since_epoch());
}

void GyroIMU::readData()
{
    txBuffer_[0] = 0x00;
    txBuffer_[1] = READ_BIT | RATE_X_LSB;

    gpioWrite(CS_GYRO,0);
    spiXfer(spiHandle_,txBuffer_,&rxBuffer_[0],2);
    spiRead(spiHandle_,&rxBuffer_[1],6); //read next 5 registers after ...
}
```

```

    xfer reads 1st, xfer order is reversed for some reason
gpioWrite(CS_GYRO,1);
sensorVals_.t = monoTimeNow_();
/*
std::cout<<"gyro reading: ";
for (int i = 0; i < 6;i++)
{
    std::cout<<std::bitset<8>(rxBuffer[i])<<", ";
}
std::cout<<std::endl;
*/
//                                LOW ...
//                                HIGH
sensorVals_.xi = (static_cast<int16_t>(rxBuffer_[0]) & 0x00FF) | ...
    (static_cast<int16_t>(rxBuffer_[2])<<8);
sensorVals_.yi = (static_cast<int16_t>(rxBuffer_[1]) & 0x00FF) | ...
    (static_cast<int16_t>(rxBuffer_[4])<<8);
sensorVals_.zi = (static_cast<int16_t>(rxBuffer_[3]) & 0x00FF) | ...
    (static_cast<int16_t>(rxBuffer_[6])<<8); //awful, switch to 8 bit ...
word length plzzzzz

sensorVals_.xf = (static_cast<float>(sensorVals_.xi)/32767.0f * ...
    range_) - offsetVals_.xf; //from datasheet
sensorVals_.yf = (static_cast<float>(sensorVals_.yi)/32767.0f * ...
    range_) - offsetVals_.yf;
sensorVals_.zf = (static_cast<float>(sensorVals_.zi)/32767.0f * ...
    range_) - offsetVals_.zf;
}

int GyroIMU::sensorInit_(int spiHandle, char* buffer)
{
    //enter normal mode
buffer[1] = 0 | (GYRO_RANGE);
buffer[0] = (0x00);//write full scale, +-2000 deg/s
range_ = 2000.0f;
//std::cout<<"bytes sent: "<<std::bitset<8>(buffer[0])<<" ...
    "<<std::bitset<8>(buffer[1])<<std::endl;// " 1: ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_GYRO,0);
spiWrite(spiHandle,buffer,2);
gpioWrite(CS_GYRO,1);

//set up low pass filter
buffer[1] = 0 | (GYRO_BANDWIDTH);
buffer[0] = (0x07);//ODR: 100, BW: 32
//buffer[0] = (0x07);//ODR: 100, BW: 32
gpioWrite(CS_GYRO,0);
spiWrite(spiHandle,buffer,2);
gpioWrite(CS_GYRO,1);
return 0;
}

void GyroIMU::calib_()
{
    const size_t calibSize = 1000;
    double calibX,calibY,calibZ = 0;

    for(size_t i = 0; i < calibSize;i++)
    {
        readData();
        calibX += sensorVals_.xf;

```

```

        calibY += sensorVals_.yf;
        calibZ += sensorVals_.zf;
    }
    offsetVals_.xf = static_cast<float>(calibX / ...
        static_cast<double>(calibSize));
    offsetVals_.yf = static_cast<float>(calibY / ...
        static_cast<double>(calibSize));
    offsetVals_.zf = static_cast<float>(calibZ / ...
        static_cast<double>(calibSize));
}

int GyroIMU::writeSingleReg()
{
    return 0;
}

void GyroIMU::populateProtobuf(dronePosVec::dronePosition& dp)
{
    dp.clear_rotation();
    dp.add_rotation(sensorVals_.xf);
    dp.add_rotation(sensorVals_.yf);
    dp.add_rotation(sensorVals_.zf);
    dp.set_timestamp_ns(sensorVals_.t.count());
}

//ACCEL-----
void AccelIMU::readData()
{
    txBuffer_[0] = 0xFF;
    txBuffer_[1] = READ_BIT | ACC_X_LSB;

    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle_,txBuffer_,2);
    spiRead(spiHandle_,rxBuffer_,6); //1st is ignored in 2nd sent byte of ...
    write
    gpioWrite(CS_ACCEL,1);
    sensorVals_.t = monoTimeNow_();
    /*
    std::cout<<"accel reading: ";
    for (int i = 0; i < 6;i++)
    {
        std::cout<<std::bitset<8>(rxBuffer[i])<<", ";
    }
    std::cout<<std::endl;
    */
    //                                LOW ...
    //                                HIGH
    sensorVals_.xi = (static_cast<int16_t>(rxBuffer_[1]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[0])<<8);
    sensorVals_.yi = (static_cast<int16_t>(rxBuffer_[3]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[2])<<8);
    sensorVals_.zi = (static_cast<int16_t>(rxBuffer_[5]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[4])<<8);

    sensorVals_.xf = (static_cast<float>(sensorVals_.xi)/32768.0f * ...
        1000.0f * std::pow(2.0,static_cast<float>(range_+1)) * 1.5) - ...
        offsetVals_.xf; //from datasheet
    sensorVals_.yf = (static_cast<float>(sensorVals_.yi)/32768.0f * ...
        1000.0f * std::pow(2.0,static_cast<float>(range_+1)) * 1.5) - ...
        offsetVals_.yf;
}

```

```

    sensorVals_.zf = (static_cast<float>(sensorVals_.zi)/32768.0f * ...
        1000.0f * std::pow(2.0, static_cast<float>(range_+1)) * 1.5) - ...
        offsetVals_.zf;
}

int AccelIMU::sensorInit_(int spiHandle, char* buffer)
{
    //enter normal mode
    buffer[1] = 0 | (ACC_PWR_CTRL);
    buffer[0] = (0x04); //enter normal mode by writing 4
    //std::cout<<"bytes sent: "<<std::bitset<8>(buffer[0])<<" ...
        "<<std::bitset<8>(buffer[1])<<std::endl;// " 1: ...
        "<<std::bitset<8>(rxBuffer[1])<<std::endl;
    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle,buffer,2);
    gpioWrite(CS_ACCEL,1);

    std::this_thread::sleep_for(std::chrono::microseconds(450)); //specified ...
        in datasheet

    //write to accelerometer range register
    buffer[1] = 0 | (ACC_RANGE);
    buffer[0] = (0x03); //+- 12g
    range_ = static_cast<float>(buffer[0]);
    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle,buffer,2);
    gpioWrite(CS_ACCEL,1);

    //set up low pass filter
    buffer[1] = 0 | (ACC_CONF);
    buffer[0] = (0x05 | (0x08<<4)); //12.5
    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle,buffer,2);
    gpioWrite(CS_ACCEL,1);
    return 0;
}

void AccelIMU::calib_()
{
    const size_t calibSize = 1000;
    double calibX = 0;
    double calibY = 0;
    double calibZ = 0;

    for(size_t i = 0; i < calibSize;i++)
    {
        readData();
        calibX += sensorVals_.xf;
        calibY += sensorVals_.yf;
        calibZ += sensorVals_.zf;
    }
    offsetVals_.xf = static_cast<float>(calibX / ...
        static_cast<double>(calibSize));
    offsetVals_.yf = static_cast<float>(calibY / ...
        static_cast<double>(calibSize));
    offsetVals_.zf = static_cast<float>(calibZ / ...
        static_cast<double>(calibSize));
}

int AccelIMU::writeSingleReg()
{

```

```
    return 0;
}

void AccelIMU::populateProtobuf(dronePosVec::dronePosition& dp)
{
    dp.clear_position();
    //dp.mutable_positiondot()->Add(sensorVals_.xf); //alt method if ...
    // needed, add_xxx should work fine however
    dp.add_position(sensorVals_.xf);
    dp.add_position(sensorVals_.yf);
    dp.add_position(sensorVals_.zf);
}
```

## G.5 Clients - Python

### G.5.1 arenaComm.py

This script needs to be in the same folder as the generated python protocol buffer code.

```
import time
import socket
import queue
import multiprocessing as mp
from . import dronePosVec_pb2
from enum import Flag, auto
import traceback #temp, for debugging
import csv

class SendrecvType(Flag):
    GENERICSEND = auto()
    GENERICRECV = auto()
    NO_MP = auto()

__global_Server_Time = 0
__interval_Send = 100000000

def getTimeNow(): #returns ns
    return time.time_ns()

#def intervalTimePrev(): #returns ns
#    return float(time.time_ns() - ((time.time_ns() - ...
#        __global_Server_Time) % __interval_Send) - __interval_Send)

class DataSending:
    mpLoop = True
    globalTimer = None
    offset = 0
    bufsize = 1024
    sendinterval = 100000000
    serverAddress = None
    nextAddress = None
    processType = None
    dataMsg = dronePosVec_pb2.dataTransfers()
    dp = dronePosVec_pb2.dronePosition()

    def __init__(self,serverAddress,processType):
        self.serverAddress = serverAddress
        self.sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        self.processType = processType
        self.sock.setblocking(True)
        global __interval_Send
        __interval_Send = self.sendinterval

    def send(self,msg):
        self.sock.send(msg)

    def recv(self):
        return self.sock.recv(self.bufsize)

    def sSyncTimer(self): #sync local timer from server, probably outdated...
        self.dataMsg.Clear()
        self.sock.settimeout(None)
        msg = self.recv()
        self.globalTimer = time.time_ns()
```

```

        self.dataMsg.ParseFromString(msg)
        syncInterval = 100000000 #100ms
        sleepLen = self.sleepTimeCalc(syncInterval, self.globalTimer)
        time.sleep(sleepLen) #sleep until sync time
        responseTime = time.time_ns()
        print("responseTime: " + str(responseTime))

        self.dataMsg.Clear()
        self.dataMsg.ID = self.processType
        self.dataMsg.timeSync_ns = responseTime
        self.dataMsg.msg = "responseTime" #probably ignored
        self.send(self.dataMsg.SerializeToString())

        self.dataMsg.ParseFromString(self.recv())
        self.offset = self.dataMsg.timeSync_ns
        self.globalTimer = self.globalTimer - self.offset
        global __global_Server_Time
        __global_Server_Time = self.globalTimer
        print("offset: " + str(self.offset))

    def sleepTimeCalc(self,interval,timer):
        return float(interval - ((time.time_ns() - timer) % ...
            interval))/1000000000.0

    def initRecv(self):
        return self.sock.recvfrom(self.buffersize)

    def dserverConnect(self): #needs to be updated, and pbstring changed
        stringRecv = None
        self.dataMsg.Clear()
        self.dataMsg.ID = self.processType
        self.dataMsg.msg = "hi"
        pbstring = self.dataMsg.SerializeToString()
        for i in range(10):
            #print("sendto: " + str(self.serverAddress))
            self.sock.sendto(pbstring,self.serverAddress) #init send
            try:
                self.sock.settimeout(3.0)
                stringRecv = self.initRecv() #initial send
                print("string received: " + str(stringRecv))
            except:
                print("timeout, retrying " + str(i))
            if not stringRecv == None:
                break
        if stringRecv == None:
            return -1 #failure to send
        print("connecting to:" + str(stringRecv[1]))
        self.sock.connect(stringRecv[1])
        #self.send(pbstring) #connection msg
        return 0 #success

    def getnextAddr(self):
        print("next address missing, getting")
        self.dataMsg.Clear()
        self.dataMsg.ID = self.processType
        self.dataMsg.type = dronePosVec_pb2.socketInfo
        self.dataMsg.msg = "socketReq"
        self.send(self.dataMsg.SerializeToString())

        self.sock.settimeout(None)
        stringRecv = self.recv()

```

```

        self.dataMsg.Clear()
        self.dataMsg.ParseFromString(stringRecv)
        print("next address: " + str(self.dataMsg.IP)+ ":" + ...
              str(self.dataMsg.port))
        return (self.dataMsg.IP,self.dataMsg.port)

    def stateChange(self):
        self.dataMsg.Clear()
        self.dataMsg.ID = self.processType
        self.dataMsg.type = dronePosVec_pb2.stateChange
        self.dataMsg.msg = "stateChangeReq"
        self.send(self.dataMsg.SerializeToString())

        self.sock.shutdown(socket.SHUT_RDWR)
        self.sock.close()

        self.sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM) ...
            #shutdown and then generate new socket
        self.sock.connect(self.nextAddress)
        self.dataMsg.Clear()
        self.dataMsg.ID = self.processType
        self.dataMsg.msg = "hi"
        self.send(self.dataMsg.SerializeToString())
        print("sucessfully connected to new socket")

    def checklist(self): #TODO: error handling
        checkList = True
        while(checkList):
            if self.globalTimer == None:
                print("timer missing, getting")
                self.dataMsg.Clear()
                self.dataMsg.ID = self.processType
                self.dataMsg.type = dronePosVec_pb2.timeSync
                self.dataMsg.msg = "syncReq"
                self.send(self.dataMsg.SerializeToString())
                self.sSyncTimer()
                continue
            elif self.nextAddress == None:
                self.nextAddress = self.getnextAddr()
            else:
                print("checkList completed, requesting state change")
                self.stateChange()
                checkList = False

    def waitForStartSignal(self):
        print("waiting for signal to start program")
        self.sock.settimeout(120)
        self.dataMsg.Clear()
        try:
            self.dataMsg.ParseFromString(self.recv())
        except Exception:
            return -1 #end program
        print(self.dataMsg.msg) #temporary
        if(self.dataMsg.type == dronePosVec_pb2.start):
            if self.dataMsg.timeSync_ns != 0: #change sending time
                self.sendinterval = self.dataMsg.timeSync_ns
                global __interval_Send
                __interval_Send = self.sendinterval #update the global ...
                interval
            return 0
        else:

```

```

        return -1

class ArenaCommunication:
    expectedTime = 0 #temp
    qSend = None
    qRecv = None
    mpEvent = mp.Event()
    pSend = None
    pRecv = None
    deviceType = None
    c = None
    dp = None
    dc = None
    posShape = []
    rotShape = []
    serverIP = None

    def __init__(self,startflags,hostname,posShape_,rotShape_,deviceType):
        self.posShape[:] = posShape_
        self.rotShape[:] = rotShape_
        self.deviceType = deviceType
        #init drone position message:
        self.dp = dronePosVec_pb2.dronePosition()
        self.dp.Clear()
        self.dp.deviceType = deviceType
        self.dp.posShape[:] = self.posShape
        self.dp.rotShape[:] = self.rotShape
        self.dc = dronePosVec_pb2.droneControl()
        self.dc.Clear()
        self.serverIP = socket.gethostbyname(hostname)

        if startflags != 0: #no start flags for manual socket starting ...
            (NO_MP reccomended)
            self.genericStarts(startflags)

    def genericStarts(self,startflags):
        if (SendrecvType.NO_MP in startflags) and ...
           ((SendrecvType.GENERICSEND in startflags) or ...
            (SendrecvType.GENERICRECV in startflags)): #exit so that ...
            multiple sockets arent created
            print("invalid flags")
            exit(-1)

        if (SendrecvType.NO_MP in startflags): # start unthreaded socket ...
            for custom writing
                self.c = DataSending(self.serverIP,self.deviceType)
                if self.c.dserverConnect() == 0:
                    self.c.checklist()
                return None

        self.qSend = mp.Queue(2)
        self.qRecv = mp.Queue(2)

        if (SendrecvType.GENERICSEND in startflags) and ...
           (SendrecvType.GENERICRECV in startflags): #starts generic ...
           socket that sends and recvs
           self.p = mp.Process(target=self.multiprocessRecvSend_,args=( ...
               (self.serverIP,20002),))
           self.p.start()
           return None

```

```

        elif (SendrecvType.GENERICSEND in startflags): # start send ...
            multiprocess only (for camera usually)
            self.p = mp.Process(target=self.multiprocessSend_, args=( ...
                (self.serverIP,20002),))
            self.p.start()
        elif (SendrecvType.GENERICRECV in startflags): # start recv ...
            multiprocess only
            self.p = mp.Process(target=self.multiprocessRecv_, args=( ...
                (self.serverIP,20002),))
            self.p.start()

    def safeQueueGet(self,_q,_block,_timeout):
        try:
            self.mpEvent.set()
            qval = _q.get(block= _block, timeout=_timeout)
            self.mpEvent.clear()
            return qval
        except Exception as e:
            print("exception:" + repr(e))
            self.mpEvent.clear()
            return str(0)

    def addPosToSendQueue(self):
        self.qSend.put(self.dp.SerializeToString())
        if (self.qSend.full() == True) and (not ...
            (self.mpEvent.is_set())): #kind of bad use of a queue because
            try:
                self.qSend.get(timeout=0.01) #empty if previous is ...
                still not read
            except Exception as e: #will be called if both threads try ...
                to get() at the same time
                print("add to queue: exception:" + repr(e))

    def addPosToRecvQueue(self):
        self.qRecv.put(self.dp.SerializeToString())
        if (self.qRecv.full() == True) and (not (self.mpEvent.is_set())):
            try:
                self.qRecv.get(timeout=0.05) #empty if previous is still ...
                not read
            except Exception as e: #will be called if both threads try to ...
                get() at the same time
                print("add to queue: exception:" + repr(e))

    def addMotToSendQueue(self):
        self.qSend.put(self.dc.SerializeToString(),True,None)
        if (self.qSend.full() == True) and (not (self.mpEvent.is_set())):
            try:
                self.qSend.get(timeout=0.01) #empty 1st in queue only, if ...
                previous is still not read
            except Exception as e: #will be called if both threads try to ...
                get() at the same time
                print("add to queue: exception:" + repr(e))

    def endmps(self): #must be called before ending program if using ...
        multiprocesses
        self.p.join()

# ----- GENERIC RECV AND SENDS

```

```

# used for inbuilt functions such as camera or if not using self made ...
    #recieve or sending methods for RL
#generic sender (used for camera)
def multiprocessSend_(self,serverAddress):
    print("sender multiprocess starting")
    self.c = DataSending(serverAddress,self.deviceType)

    qtimeout = 10
    if self.c.dserverConnect() == 0:
        self.c.checklist()
        if self.c.waitForStartSignal() == -1:
            self.c.mpLoop = False #dont start program

    #main loop
    while(self.c.mpLoop):
        time.sleep(self.c.sleepTimeCalc( ...
                    self.c.sendinterval,self.c.globalTimer))
        try:
            #c.send(msg) #test
            self.c.send(self.c.safeQueueGet(self.qSend,True,qtimeout))
            qtimeout = 5 # set to 5s after initial
        except Exception as e:
            self.c.mpLoop = False #timeout
            print("exception:" + repr(e))
            break

    print("sender multiprocess closing")

#generic reciever
def multiprocessRecv_(self,serverAddress):
    print("reciever multiprocess starting")
    self.c = DataSending(serverAddress,self.deviceType)

    recvMsg = 0
    if self.c.dserverConnect() == 0:
        self.c.checklist()
        if self.c.waitForStartSignal() == -1:
            self.c.mpLoop = False #dont start program

    #main loop
    self.sock.settimeout(10.0) #long initial timeout for if server ...
        #is still doing things
    while(self.c.mpLoop):
        time.sleep(self.c.sleepTimeCalc( ...
                    self.c.sendinterval,self.c.globalTimer))
        try:
            recvMsg = self.c.recv()
            if (self.qSend.full() == True) and (not ...
                (self.mpEvent.is_set())):
                try:
                    self.qSend.get(timeout=0.05) #empty if ...
                        previous is still not read
                except Exception: #will be called if both threads ...
                    try to get() at the same time
                    pass
            self.qSend.put(recvMsg)

            self.sock.settimeout(5.0)
        except Exception as e:
            self.c.mpLoop = False #timeout
            print("exception:" + repr(e))

```

```

        break

    print("reciever multiprocess closing")

    #generic synchronised send and reciever
def multiprocessRecvSend_(self,serverAddress):
    print("send and recieve multiprocess starting")
    self.c = DataSending(serverAddress,self.deviceType)
    #csv
    fields = ['time diff']
    filename = "times.csv"
    #csv
    with open(filename, 'w') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(fields)

    qtimeout = 10
    recvMsg = []
    if self.c.dserverConnect() == 0:
        self.c.checklist()
        if self.c.waitForStartSignal() == -1:
            self.c.mpLoop = False #dont start program

    #main loop
    self.c.sock.settimeout(10.0) #long initial timeout for if ...
                                #server is still doing things
    while(self.c.mpLoop):
        self.expectedTime = (self.c.sleepTimeCalc( ...
                                self.c.sendinterval,self.c.globalTimer) ...
                                *1000000000.0) + getTimeNow()

        print(self.c.sleepTimeCalc( ...
                                self.c.sendinterval,self.c.globalTimer))
        time.sleep(self.c.sleepTimeCalc( ...
                                self.c.sendinterval,self.c.globalTimer))
        #self.expectedTime = getTimeNow()
        try:
            self.c.send(self.safeQueueGet(self.qSend,True, ...
                                         qtimeout))
            recvMsg = self.c.recv()
            if self.qRecv.full() == True:
                try:
                    self.qRecv.get(timeout=0.05) #empty if ...
                                                #previous is still not read
                except Exception: #will be called if both ...
                                    #threads try to get() at the same time
                                    pass
            self.qRecv.put(recvMsg)
            writer.writerow({(getTimeNow() - ...
                            self.expectedTime)/1000000})
            print("msg recieived with time difference (ms): " + ...
                  str((getTimeNow() - self.expectedTime)/1000000))
            self.c.sock.settimeout(5.0)
            qtimeout = 5 # set to 5s after
        except Exception as e:
            self.c.mpLoop = False #timeout
            print("exception:" + repr(e))
            break

    print("send and reciever multiprocess closing")

```

### G.5.2 RLsendRecvEx.py

```
from Modules import arenaComm
from Modules import dronePosVec_pb2
import math
import time
import traceback

hostname = 'b12.uia.no'
messager = arenaComm.ArenaCommunication(arenaComm.SendrecvType.GENERICRECV ...
    | arenaComm.SendrecvType.GENERICSEND, ...
    hostname, (1, 3), (3, 3), dronePosVec_pb2.rl)

msg = "0" #string
motorvals = 0
running = True
dp = dronePosVec_pb2.dronePosition()

try:
    while running:
        if messager.qRecv.empty() == False:
            msg = messager.safeQueueGet(messager.qRecv, True, 5)
            if not (len(msg) < 2): #a string with 0 might get sent due to ...
                an error
            dp.ParseFromString(msg)
            #print("msg received: " + str(dp.position))
        #else:
            #print("fail msg received")
        #
        # PYTORCH COULD GO HERE FOR EXAMPLE
        #
        messager.dc.Clear()
        motorvals = math.sin(math.pi*2*((time.time()%20)/20)) #make a sine ...
            wave function with a period of 20s
        messager.dc.motorFL = motorvals
        messager.dc.motorFR = motorvals
        messager.dc.motorBL = motorvals
        messager.dc.motorBR = motorvals
        messager.dc.killswitch = False
        messager.addMotToSendQueue()
except KeyboardInterrupt: #exit with ctrl+c
    pass

#end multiprocesses
print("\n ending program")
messager.endmps()
```

## G.6 Arena

### G.6.1 bmi088.cpp

```
#include <iostream>
#include <pigpio.h>
#include <cstring>
#include <bitset>
#include <thread>
#include <chrono>
#include <bit>

#define LED 18 //GPIO 18, physical pin 12
#define CS_ACCEL 12 //GPIO 12, physical pin 22
#define CS_GYRO 13
#define SPI_CHANNEL 1

#define ACC_PWR_CTRL 0x7D

int main()
{
    if(gpioInitialise()<0) // must be initialized first
    {
        std::cout<<"initialisation failed";
        return -1;
    }
    const size_t buffersize = 16;
    char txBuffer[buffersize];
    char rxBuffer[buffersize];
    memset(rxBuffer,3,buffersize); //set to known value (0000 0011)
    memset(txBuffer,0,buffersize); //set to known value (0000 0011)
    int spiHandle = ...
        spiOpen(SPI_CHANNEL,1000000,(0x0<<15)|(0x0<<14)|(0x10<<16)| ...
        (0x1<<8) | (0x1<<5) | (0x1<<6) | (0x1<<7) | (0x0)); // using mode 0, ...
        disable ux as cs, enable auxilarary spi, make 16 bit word length
    std::cout<<"spiHandle:" <<spiHandle<<std::endl;
    if (spiHandle<0)
    {
        std::cout<<"spi open failed";
        return -1;
    }
    bool running = true;
    int a = 0;

    gpioSetMode(LED, PI_OUTPUT);
    gpioSetMode(CS_ACCEL, PI_OUTPUT);
    gpioSetMode(CS_GYRO, PI_OUTPUT);
//-----MAIN-----
    std::this_thread::sleep_for(std::chrono::milliseconds(1)); //wait 1ms ...
        after startup
    gpioWrite(CS_ACCEL,1);
    gpioWrite(CS_GYRO,1);
    gpioWrite(LED,0);

    //enter normal mode
    txBuffer[1] = 0 | (ACC_PWR_CTRL);
    txBuffer[0] = (0x04); //enter normal mode by writing 4
    std::cout<<"bytes sent: "<<std::bitset<8>(txBuffer[0])<<" ...
        "<<std::bitset<8>(txBuffer[1])<<std::endl;// " 1: ...
        "<<std::bitset<8>(rxBuffer[1])<<std::endl;
```

```

gpioWrite(LED,1);
gpioWrite(CS_ACCEL,0);
spiWrite(spiHandle,txBuffer,2);
gpioWrite(CS_ACCEL,1);

std::this_thread::sleep_for(std::chrono::milliseconds(1));
//read ACC_PWR_CTRL to confirm
txBuffer[1] = (1<<7) | (ACC_PWR_CTRL);
txBuffer[0] = (0xFF);
std::cout<<"bytes read: "<<std::bitset<8>(txBuffer[0])<<" ...
    "<<std::bitset<8>(txBuffer[1])<<std::endl;// " 1: ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_ACCEL,0);
spiWrite(spiHandle,txBuffer,2);
spiRead(spiHandle,rxBuffer,2);
std::cout<<"ACC_PWR_CTRL: "<<std::bitset<8>(rxBuffer[1])<<std::endl;// ...
    " 1: "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_ACCEL,1);

//read gyro
txBuffer[1] = (1<<7) | (0x00);
txBuffer[0] = (0xFF);
gpioWrite(CS_GYRO,0);
spiXfer(spiHandle,txBuffer,rxBuffer,2);
std::cout<<"Gyro ID: "<<std::bitset<8>(rxBuffer[0])<<" ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;//first byte is ONLY ...
        ignored on ACCEL, NOT GYRO
gpioWrite(CS_GYRO,1);

gpioWrite(CS_ACCEL,0);
spiWrite(spiHandle,txBuffer,2);
spiRead(spiHandle,rxBuffer,2);
std::cout<<"Accel ID: "<<std::bitset<8>(rxBuffer[0])<<" ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;// " 1: ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_ACCEL,1);
gpioWrite(LED,0);

spiClose(spiHandle);
gpioTerminate(); // call when done with library
return 0;
}

```

## G.6.2 i2cAS5600.cpp

```
#include <iostream>
#include <bitset>
#include <pigpio.h>

#define LED 18 //GPIO 18, pin 12

//#define SCL 3 //pin 3
//#define SDA 2 //pin 5
const unsigned int addr = 0x36;
const unsigned int bus = 1;
int handle;

void i2cinit()
{
    handle = i2cOpen(bus, addr, 0);
    if(handle<0)
    {
        std::cout<<"opening i2c failed"<<std::endl;
    }
    else
    {
        std::cout<<"handle: " <<handle<<std::endl;
    }
}

unsigned int readRegisterByte(const uint8_t& regAddrH,const uint8_t& regAddrL)
{

    unsigned int H = static_cast<unsigned ...>(i2cReadByteData(handle,regAddrH)); //reads H byte (11:8)
    H &= 0x000F; //HL = 12 bit value, removes all possible 1s in H ...
    before last 4 bits
    unsigned int L = static_cast<unsigned ...>(i2cReadByteData(handle,regAddrL)); //reads L byte (7:0)
    unsigned int combine = 0 | L | (H<<8);
    return combine;
}

unsigned int readRegisterByte(const uint8_t& regAddr)
{
    return i2cReadByteData(handle,regAddr); //reads a single register ...
    in the i2c device
}

int main()
{

    if(gpioInitialise()<0) // must be initialized first
    {
        std::cout<<"initialisation failed";
        return 0;
    }

    i2cinit();

    int cinput;
    bool loop = true;
    while(loop)
    {
```

```

        std::cout<<"write input:";
        std::cin >> cinput;
        std::cout<<" input: "<<cinput<<std::endl;
        if (cinput == 0)
        {
            loop = false;
            std::cout<<"exiting"<<std::endl;
            gpioWrite(LED,0);
        }
        else if(cinput == 1)
        {
            std::cout<<"reading status register: " ...
                <<std::bitset<8>(readRegisterByte(0x0B)) ...
                <<std::endl;
        }
        else if(cinput == 2)
        {
            std::cout<<"reading H and L: " ...
                <<readRegisterByte(0x0E,0x0F)<<std::endl;
        }
        else if (cinput == 3)
        {
            std::cout<<"mass reading:"<<std::endl;
            unsigned int reading;
            for(int i = 0; i <= 50000; i++)
            {
                reading = readRegisterByte(0x0E,0x0F);
                std::cout<<std::bitset<16>(reading);
                std::cout<<" rotation: " << ...
                    (static_cast<float>(reading)/4096.0) * ...
                    360.0<<"\r";
            }
            std::cout<<std::endl;
        }
    }
    else
    {
        //std::cout<<"reading i2c: " << ...
        readRegisterByte()<<std::endl;
    }
}

gpioTerminate(); // call when done with library
return 0;
}

```

### G.6.3 azionly.py

This code is modified from a code in the 2022 Drone Arena rapport, originally created by Martin Økter [14].

```
import signal
import sys
import RPi.GPIO as GPIO
import time
import matplotlib.pyplot as plt

#Setup GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

y_list = [0]
x_list = [0]

class pwm_motor:
    def __init__(self, motor_pin, freq, C1_pin, C2_pin, AIN1_pin, AIN2_pin):
        self.C1 = C1_pin
        self.C2 = C2_pin
        self.AIN1 = AIN1_pin
        self.AIN2 = AIN2_pin
        self.C1_count = 0
        self.C2_count = 0

        self.hall_per_round = 300

        self.curr_ang = 0
        self.last_error = 0
        self.i_last = 0

        self.last_time = time.time() #Set a start time as a setpoint
        self.dir = 0

        GPIO.setup(motor_pin, GPIO.OUT)
        GPIO.setup(self.AIN1, GPIO.OUT)
        GPIO.setup(self.AIN2, GPIO.OUT)

        #Setup hall interrupt
#GPIO.setup(self.C1, GPIO.IN, pull_up_down=GPIO.PUD_UP)
#    GPIO.setup(self.C2, GPIO.IN, pull_up_down=GPIO.PUD_UP)
#    GPIO.add_event_detect(self.C1, GPIO.RISING, callback=self.hall_callback)
#    GPIO.add_event_detect(self.C2, GPIO.RISING, ...
#callback=self.hall_callback)

        self.motor = GPIO.PWM(motor_pin, freq)
        self.motor.start(0)

    def pwm_stop(self):
        self.motor.stop() #Stop all movement

    def runPWM(self, moveDir):
        if moveDir == True: # clockwise
            GPIO.output(self.AIN1, GPIO.HIGH)
            GPIO.output(self.AIN2, GPIO.LOW)
        else: #ccw
            GPIO.output(self.AIN1, GPIO.LOW)
            GPIO.output(self.AIN2, GPIO.HIGH)
```

```

        self.motor.ChangeDutyCycle(25)
        time.sleep(3)
        self.motor.ChangeDutyCycle(0)

#Function setup
def signal_handler(sig, frame):
    global y_list
    global x_list
    fig, ax = plt.subplots()
    ax.grid()
    ax.set_xlabel("Iteration")
    ax.set_ylabel("Angle")
    ax.plot(x_list, y_list)
    plt.show()
    GPIO.cleanup()
    sys.exit(0)

#Initialize motors
#Initial basetime 0.0208
mainRotMot = pwm_motor(12,2000,25,16,27,17)

#mainStepper.go_rounds(1,'Down','Up',1234567890)
while True:
    print('Initiate run')
    mainRotMot.runPWM(False)
    print('Run complete')
    #mainRotMot.pwm_stop()
    #signal.signal(signal.SIGINT, signal_handler)
    #signal.pause()
    time.sleep(10)
mainRotMot.pwm_stop()

```

#### G.6.4 wiringpitest.cpp

```
#include <iostream>
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <cstring>
#include <set>
#include <thread>
#include <chrono>

#define SPI_CHAN 0
#define LED 1 //wiringpi 1, pin 12
#define CS_PIN 6//wiring pi 6, pin 22

int main()
{
    int myFd;
    wiringPiSetup();
    if(myFd = wiringPiSPISetup(SPI_CHAN, 1000000)<0) // must be ...
        initialized first
    {
        std::cout<<"initialisation failed";
        return -1;
    }
    const size_t buffersize = 8;
    unsigned char txBuffer[buffersize];
    //char txBuffer[buffersize];
    //memset(txBuffer,3,buffersize); //set to known value (0000 0011)
    memset(txBuffer,0,buffersize); //set to known value (0000 0011)

    bool running = true;
    int a = 0;

    pinMode(LED, OUTPUT);
    pinMode(CS_PIN, OUTPUT);
    //-----MAIN-----
    digitalWrite(CS_PIN,1);
    digitalWrite(LED,0);

    //-----PWR_MGMT_1 H_RESET -----
    txBuffer[0] = 0 | 0x6B;
    txBuffer[1] = (0 | (1<<7) | (1<<2));
    if(wiringPiSPIDataRW(SPI_CHAN,txBuffer,2) == -1)
    {
        std::cout<<"failed to write to spi"<<std::endl;
        return -1;
    }
    //-----disable i2c-----
    txBuffer[0] = 0 | 0x6A;
    txBuffer[1] = (0 | (1<<4));
    std::this_thread::sleep_for(std::chrono::milliseconds(11));
    std::cout<<wiringPiSPIDataRW(SPI_CHAN,txBuffer,2)<<std::endl;

    while(running)
    {
        //test: reading WHO_AM_I register

        for(int i = 0; i < buffersize; i++)
        {
            txBuffer[i] = (1<<7) | 0x71;
        }
    }
}
```

```

    }

txBuffer[1] = ~0;

std::cout<<"tx: ";
for(int i = 0; i < buffersize; i++)
{
    std::cout<<std::bitset<8>(txBuffer[i])<<" ";
}
std::cout<<std::endl;

digitalWrite(CS_PIN,0);

//std::cout<<"write: "<< spiWrite(spiHandle,txBuffer,1)<<" msg: ...
    << std::bitset<8>(txBuffer[0]) <<std::endl;
//spiRead(spiHandle,txBuffer,buffersize);

std::cout<<"xfer: "<< wiringPiSPIDataRW(SPI_CHAN,txBuffer,8) ...
    <<std::endl;

digitalWrite(CS_PIN,1);

std::cout<<"rx: ";
for(int i = 0; i < buffersize; i++)
{
    std::cout<<std::bitset<8>(txBuffer[i])<<" ";
}
std::cout<<std::endl;

std::cin>>a;
if (a == 0)
{
    running = false;
}
}

return 0;
}

```

## G.7 Drone

### G.7.1 IMU test reading

```
#include <iostream>
#include <pigpio.h>
#include <cstring>
#include <bitset>
#include <thread>
#include <chrono>
#include <bit>

#define CS_ACCEL 8 //GPIO 12, physical pin 22
#define CS_GYRO 7
#define SPI_CHANNEL 0

#define ACC_PWR_CTRL 0x7D

#define READ_BIT 0x01

int main()
{
    if(gpioInitialise()<0) // must be initialized first
    {
        std::cout<<"initialisation failed";
        return -1;
    }
    const size_t buffersize = 16;
    char txBuffer[buffersize];
    char rxBuffer[buffersize];
    memset(rxBuffer,3,buffersize); //set to known value (0000 0011)
    memset(txBuffer,0,buffersize); //set to known value (0000 0011)
    int spiHandle = ...
        spiOpen(SPI_CHANNEL,1000000,(0x0<<15)|(0x0<<14)|(0x10<<16)| ...
            (0x0<<8) | (0x1<<5) | (0x1<<6) | (0x1<<7) | (0x0)); // using mode 0, ...
        disable ux as cs, enable auxilarary spi, make 16 bit word length
    std::cout<<"spiHandle:" <<spiHandle<<std::endl;
    if (spiHandle<0)
    {
        std::cout<<"spi open failed";
        return -1;
    }
    bool running = true;
    int a = 0;

    gpioSetMode(CS_ACCEL, PI_OUTPUT);
    gpioSetMode(CS_GYRO, PI_OUTPUT);
    //gpioSetMode(CS_PIN, PI_OUTPUT);
    //-----MAIN-----
    std::this_thread::sleep_for(std::chrono::milliseconds(1)); //wait 1ms ...
        after startup
    //gpioWrite(CS_PIN,1); //to disable previous IMU
    gpioWrite(CS_ACCEL,1);
    gpioWrite(CS_GYRO,1);

    //enter normal mode
    txBuffer[0] = 0 | (ACC_PWR_CTRL);
    txBuffer[1] = (0x04); //enter normal mode by writing 4
    std::cout<<"bytes sent: "<<std::bitset<8>(txBuffer[0])<<" ...
        "<<std::bitset<8>(txBuffer[1])<<std::endl; // " 1: ...
    
```

```

    "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_ACCEL,0);
spiWrite(spiHandle,txBuffer,2);
gpioWrite(CS_ACCEL,1);

std::this_thread::sleep_for(std::chrono::milliseconds(1));
//read ACC_PWR_CTRL to confirm
txBuffer[0] = (1<<7) | (ACC_PWR_CTRL);
txBuffer[1] = (0xFF);
std::cout<<"bytes read: "<<std::bitset<8>(txBuffer[0])<<" ...
    "<<std::bitset<8>(txBuffer[1])<<std::endl;// " 1: ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_ACCEL,0);
spiWrite(spiHandle,txBuffer,2);
spiRead(spiHandle,rxBuffer,2);
std::cout<<"ACC_PWR_CTRL: "<<std::bitset<8>(rxBuffer[1])<<std::endl;// ...
    " 1: "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_ACCEL,1);

//read gyro
txBuffer[0] = (1<<7) | (0x00);
txBuffer[1] = (0xFF);
gpioWrite(CS_GYRO,0);
spiXfer(spiHandle,txBuffer,rxBuffer,2);
std::cout<<"Gyro ID: "<<std::bitset<8>(rxBuffer[0])<<" ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;//first byte is ONLY ...
        ignored on ACCEL, NOT GYRO
gpioWrite(CS_GYRO,1);

gpioWrite(CS_ACCEL,0);
spiWrite(spiHandle,txBuffer,2);
spiRead(spiHandle,rxBuffer,2);
std::cout<<"Accel ID: "<<std::bitset<8>(rxBuffer[0])<<" ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;// " 1: ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_ACCEL,1);

spiClose(spiHandle);
gpioTerminate(); // call when done with library
return 0;
}

```

## G.7.2 IMU only code

This section includes code used for the results for the IMU, it is slightly more updated than the drone client code for the IMU sections, and uses the main SPI0 rather than the auxiliary SPI1 which the drone client code uses.

Within the directory is a cmake/Modules/Findpigpio.cmake file, which is included in the pigpio library.

### G.7.2.1 CmakeListst.txt

```
cmake_minimum_required(VERSION 3.8)
project(droneClient)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
    add_compile_options(-Wall -Wextra -Wpedantic)
endif()

#uncomment for debugging:
#set(CMAKE_BUILD_TYPE DEBUG)
#set(CMAKE_CXX_FLAGS_DEBUG "${CMAKE_CXX_FLAGS_DEBUG} -g")
#set(CMAKE_CXX_FLAGS "-g")

find_package(Protobuf 3.12.4 REQUIRED)
#find_package(pigpio REQUIRED)

list(APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake/Modules)

include_directories(${Protobuf_INCLUDE_DIRS})
include_directories(${HOME}/.local/include)

add_executable(imuDrone src imuMainTest.cpp src imuImplementation.cpp)
target_link_libraries(imuDrone pigpio)

#target_include_directories(imuDrone PUBLIC
#${<BUILD_INTERFACE>:${CMAKE_CURRENT_SOURCE_DIR}/include}
#${<INSTALL_INTERFACE>:include})

install(TARGETS
imuDrone
DESTINATION lib/${PROJECT_NAME})
```

### G.7.2.2 droneIMU.h

```
#ifndef DRONEIMU_H
#define DRONEIMU_H

#include <thread>
#include <chrono>

//pin defs
//#define LED 18 //GPIO 18, physical pin 12
#define CS_ACCEL 8 //GPIO 12, physical pin 22
#define CS_GYRO 7
//spi flags:
#define SPI_CHANNEL 0
#define SPI_SPEED 1000000
#define SPI_MODE_0 0x0
#define SPI_WORDLEN_16 (0x10<<16)
#define SPI_DISABLE_UX ((0x1<<5) | (0x1<<6) | (0x1<<7))
#define SPI_AUXILIARY (0x1<<8)

//registers:
#define READ_BIT (1<<7)
#define ACC_PWR_CTRL 0x7D
#define ACC_X_LSB 0x12
#define ACC_RANGE 0x41
#define GYRO_RANGE 0x0F
#define RATE_X_LSB 0x02
#define GYRO_BANDWIDTH 0x10
#define ACC_CONF 0x40

void gpioEnd(int spiHandle);
int spiInit(); //push to class but leave for now

using ns_t = std::chrono::nanoseconds;

struct outputVals
{
    double xf,yf,zf = 0;
    int16_t xi,yi,zi = 0;
    ns_t t;
};

class IIMU
{
public:
    IIMU(int handle)
        :spiHandle_(handle)
    {
    }

    //virtual void readData(int spiHandle,char* txBuffer,char* rxBuffer, ...
    //    struct outputVals& vals) = 0 ;
    virtual int writeSingleReg() = 0;
    //virtual void populateProtobuf(dronePosVec::dronePosition& dp) = 0;
    outputVals sensorVals;

    void numIntergration(outputVals& valsPrev, outputVals& valsOut);
    void dubnumIntergration(outputVals& valsPrev, outputVals& singVals, ...
        outputVals& dubvalsOut);

protected:
```

```

        outputVals offsetVals_;
    virtual int sensorInit_(int spiHandle, char* buffer) = 0;
    ns_t monoTimeNow_();
    virtual void calib_() = 0;

    const int spiHandle_;
    float range_;
    const size_t bufferSize = 16;
    char txBuffer_[16];
    char rxBuffer_[16];

}; //IIMU

class GyroIMU : public IIMU
{
public:
    GyroIMU(int handle)
    :IIMU(handle)
    {
        sensorInit_(spiHandle_,txBuffer_);
        //calib_();
    }
    void readData();
    virtual int writeSingleReg() override;
    using IIMU::numIntergration;
    //virtual void populateProtobuf(dronePosVec::dronePosition& dp) override;

protected:
    virtual int sensorInit_(int spiHandle, char* buffer) override;
    virtual void calib_() override;

}; //GyroIMU

class AccelIMU : public IIMU
{
public:
    AccelIMU(int handle)
    :IIMU(handle)
    {
        sensorInit_(spiHandle_,txBuffer_);
        //calib_();
    }
    void readData();
    virtual int writeSingleReg() override;
    using IIMU::numIntergration;
    //virtual void populateProtobuf(dronePosVec::dronePosition& dp) override;

protected:
    virtual int sensorInit_(int spiHandle,char* buffer) override;
    virtual void calib_() override;

}; //AccelIMU

#endif //DRONEIMU_H

```

### G.7.2.3 imuImplementation.cpp

```
#include "droneIMU.h"
#include <pigpio.h>
#include <iostream>
#include <cmath>
#include <bitset>

void gpioEnd(int spiHandle)
{
    spiClose(spiHandle);
    gpioTerminate(); // call when done with library
}

int spiInit()
{
    if(gpioInitialise()<0) // must be initialized first
    {
        std::cout<<"initialisation failed";
        return -1;
    }
    int spiHandle = spiOpen(SPI_CHANNEL,SPI_SPEED, SPI_DISABLE_UX | ...
                           SPI_MODE_0); // using mode 0, disable ux as cs, enable auxilarary ...
                           // spi, make 16 bit word length
    //int spiHandle = spiOpen(SPI_CHANNEL,SPI_SPEED,SPI_AUXILIARY | ...
                           // SPI_DISABLE_UX | SPI_MODE_0); // using mode 0, disable ux as cs, ...
                           // enable auxilarary spi, make 16 bit word length
    if (spiHandle<0)
    {
        std::cout<<"spi open failed";
        return -2;
    }
    //gpioSetMode(LED, PI_OUTPUT);
    gpioSetMode(CS_ACCEL, PI_OUTPUT);
    gpioSetMode(CS_GYRO, PI_OUTPUT);

    gpioWrite(CS_ACCEL,1);
    gpioWrite(CS_GYRO,1);
    //gpioWrite(LED,0); //led for any debugging, TODO: should be removed ...
                      // later though

    std::this_thread::sleep_for(std::chrono::milliseconds(1)); //wait 1ms ...
                                                               // after startup

    return spiHandle;
}

void IIMU::numIntergration(outputVals& valsPrev, outputVals& valsOut)
{
    double dt = static_cast<double>((sensorVals.t - valsPrev.t).count()) / ...
               1000000000.0f;

    valsOut.xf += (sensorVals.xf - valsPrev.xf) * dt;
    valsOut.yf += (sensorVals.yf - valsPrev.yf) * dt;
    valsOut.zf += (sensorVals.zf - valsPrev.zf) * dt;
}

void IIMU::dubnumIntergration(outputVals& valsPrev, outputVals& singVals, ...
                               outputVals& dubvalsOut)
{
```

```

    double dt = static_cast<double>((sensorVals.t - valsPrev.t).count()) / ...
        1000000000.0f;

    dubvalsOut.xf += (singVals.xf - valsPrev.xf) * dt;
    dubvalsOut.yf += (singVals.yf - valsPrev.yf) * dt;
    dubvalsOut.zf += (singVals.zf - valsPrev.zf) * dt;
}

ns_t IIMU::monoTimeNow_()
{
    return ...
    std::chrono::duration_cast<ns_t>(std::chrono::steady_clock::now( ...
        ).time_since_epoch());
}

void GyroIMU::readData()
{
    txBuffer_[1] = 0xFF;
    txBuffer_[0] = READ_BIT | RATE_X_LSB;

    gpioWrite(CS_GYRO,0);
    spiWrite(spiHandle_,txBuffer_,1);
    //spiXfer(spiHandle_,txBuffer_,rxBuffer_,2);
    //std::cout<<std::bitset<8>(rxBuffer_[1])<<std::endl;
    //rxBuffer_[0] = rxBuffer_[1];
    spiRead(spiHandle_,&rxBuffer_[0],6); //read next 5 registers
    //spiRead(spiHandle_,&rxBuffer_[1],5); //read next 5 registers
    gpioWrite(CS_GYRO,1);
    sensorVals.t = monoTimeNow_();

    /*
    std::cout<<"gyro reading: ";
    for (int i = 0; i < 6;i++)
    {
        std::cout<<std::bitset<8>(rxBuffer_[i])<<", ";
    }
    std::cout<<"\r";
    */

    //                                LOW ...
    //                                HIGH
    sensorVals.xi = (static_cast<int16_t>(rxBuffer_[0]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[1])<<8);
    sensorVals.yi = (static_cast<int16_t>(rxBuffer_[2]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[3])<<8);
    sensorVals.zi = (static_cast<int16_t>(rxBuffer_[4]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[5])<<8);

    sensorVals.xf = (static_cast<double>(sensorVals.xi)/32767.0f * ...
        range_); // - offsetVals_.xf; //from datasheet
    sensorVals.yf = (static_cast<double>(sensorVals.yi)/32767.0f * ...
        range_); // - offsetVals_.yf;
    sensorVals.zf = (static_cast<double>(sensorVals.zi)/32767.0f * ...
        range_); // - offsetVals_.zf;
}

int GyroIMU::sensorInit_(int spiHandle, char* buffer)
{
    //enter normal mode
    buffer[0] = 0 | (GYRO_RANGE);
}

```

```

buffer[1] = (0x01); //write full scale, +-1000 deg/s
range_ = 1000.0f;
//std::cout<<"bytes sent: "<<std::bitset<8>(buffer[0])<<" ...
    "<<std::bitset<8>(buffer[1])<<std::endl;// " 1: ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;
gpioWrite(CS_GYRO,0);
spiWrite(spiHandle,buffer,2);
gpioWrite(CS_GYRO,1);

//set up low pass filter
buffer[0] = 0 | (GYRO_BANDWIDTH);
buffer[1] = (0x02); //ODR: 116
//buffer[0] = (0x07); //ODR: 100, BW: 32
gpioWrite(CS_GYRO,0);
spiWrite(spiHandle,buffer,2);
gpioWrite(CS_GYRO,1);

//read id
txBuffer_[0] = (1<<7) | (0x00);
txBuffer_[1] = (0xFF);
gpioWrite(CS_GYRO,0);
spiXfer(spiHandle,txBuffer_,rxBuffer_,2);
std::cout<<"Gyro ID: "<<std::bitset<8>(rxBuffer_[0])<<" ...
    "<<std::bitset<8>(rxBuffer_[1])<<std::endl;//first byte is ONLY ...
        ignored on ACCEL, NOT GYRO
gpioWrite(CS_GYRO,1);

//self test
txBuffer_[0] = 0 | (0x3C);
txBuffer_[1] = (0x01);
gpioWrite(CS_GYRO,0);
spiWrite(spiHandle,txBuffer_,2);
gpioWrite(CS_GYRO,1);

std::this_thread::sleep_for(std::chrono::microseconds(450)); //specified ...
    in datasheet

txBuffer_[0] = READ_BIT | (0x3C);
txBuffer_[1] = (0xFF);
gpioWrite(CS_GYRO,0);
spiXfer(spiHandle,txBuffer_,rxBuffer_,2);
std::cout<<"Gyro self test: "<<std::bitset<8>(rxBuffer_[0])<<" ...
    "<<std::bitset<8>(rxBuffer_[1])<<std::endl;//first byte is ONLY ...
        ignored on ACCEL, NOT GYRO
gpioWrite(CS_GYRO,1);

txBuffer_[0] = 0 | (0x3C);
txBuffer_[1] = (0x00);
gpioWrite(CS_GYRO,0);
spiWrite(spiHandle,txBuffer_,2);
gpioWrite(CS_GYRO,1);

return 0;
}

void GyroIMU::calib_()
{
    const size_t calibSize = 1000;
    double calibX,calibY,calibZ = 0;

    for(size_t i = 0; i < calibSize;i++)

```

```

    {
        readData();
        calibX += sensorVals.xf;
        calibY += sensorVals.yf;
        calibZ += sensorVals.zf;
    }
    offsetVals_.xf = static_cast<double>(calibX / ...
        static_cast<double>(calibSize));
    offsetVals_.yf = static_cast<double>(calibY / ...
        static_cast<double>(calibSize));
    offsetVals_.zf = static_cast<double>(calibZ / ...
        static_cast<double>(calibSize));
}

int GyroIMU::writeSingleReg()
{
    return 0;
}
/*
void GyroIMU::populateProtobuf(dronePosVec::dronePosition& dp)
{
    dp.clear_rotation();
    dp.add_rotation(sensorVals.xf);
    dp.add_rotation(sensorVals.yf);
    dp.add_rotation(sensorVals.zf);
    dp.set_timestamp_ns(sensorVals.t.count());
}
*/
//ACCEL-----

void AccelIMU::readData()
{
    txBuffer_[1] = 0xFF;
    txBuffer_[0] = READ_BIT | ACC_X_LSB;

    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle_,txBuffer_,2);
    spiRead(spiHandle_,rxBuffer_,6); //1st is ignored in 2nd sent byte of ...
        write
    gpioWrite(CS_ACCEL,1);
    sensorVals.t = monoTimeNow_();

    /*
    std::cout<<"accel reading: ";
    for (int i = 0; i < 6;i++)
    {
        std::cout<<std::bitset<8>(rxBuffer_[i])<<, " ;
    }
    std::cout<<std::endl;
    */

    //                                LOW ...                                HIGH
    sensorVals.xi = (static_cast<int16_t>(rxBuffer_[0]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[1])<<8);
    sensorVals.yi = (static_cast<int16_t>(rxBuffer_[2]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[3])<<8);
    sensorVals.zi = (static_cast<int16_t>(rxBuffer_[4]) & 0x00FF) | ...
        (static_cast<int16_t>(rxBuffer_[5])<<8);
}

```

```

sensorVals.xf = (static_cast<double>(sensorVals.xi)/32768.0f * 1000.0f ...
    * std::pow(2.0, static_cast<double>(range_+1)) * 1.5) - ...
    offsetVals_.xf; //from datasheet
sensorVals.yf = (static_cast<double>(sensorVals.yi)/32768.0f * 1000.0f ...
    * std::pow(2.0, static_cast<double>(range_+1)) * 1.5) - offsetVals_.yf;
sensorVals.zf = (static_cast<double>(sensorVals.zi)/32768.0f * 1000.0f ...
    * std::pow(2.0, static_cast<double>(range_+1)) * 1.5) - offsetVals_.zf;
}

int AccelIMU::sensorInit_(int spiHandle, char* buffer)
{
    //enter normal mode
    buffer[0] = 0 | (ACC_PWR_CTRL);
    buffer[1] = (0x04); //enter normal mode by writing 4
    //std::cout<<"bytes sent: "<<std::bitset<8>(buffer[0])<<" ...
    "<<std::bitset<8>(buffer[1])<<std::endl;// " 1: ...
    "<<std::bitset<8>(rxBuffer[1])<<std::endl;
    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle,buffer,2);
    gpioWrite(CS_ACCEL,1);

    std::this_thread::sleep_for(std::chrono::microseconds(450)); ...
    //specified in datasheet

    //write to accelerometer range register
    buffer[0] = 0 | (ACC_RANGE);
    buffer[1] = (0x03); //+- 12g
    range_ = static_cast<double>(buffer[1]);
    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle,buffer,2);
    gpioWrite(CS_ACCEL,1);

    //set up low pass filter
    buffer[0] = 0 | (ACC_CONF);
    buffer[1] = (0x08 | (0x0A<<4)); //100, normal
    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle,buffer,2);
    gpioWrite(CS_ACCEL,1);

    //read accel ID
    txBuffer_[0] = (1<<7) | (0x00);
    txBuffer_[1] = (0xFF);

    gpioWrite(CS_ACCEL,0);
    spiWrite(spiHandle,txBuffer_,2);
    spiRead(spiHandle,rxBuffer_,2);
    std::cout<<"Accel ID: "<<std::bitset<8>(rxBuffer_[0])<<" ...
    "<<std::bitset<8>(rxBuffer_[1])<<std::endl;
    gpioWrite(CS_ACCEL,1);

    return 0;
}

void AccelIMU::calib_()
{
    const size_t calibSize = 1000;
    double calibX = 0;
    double calibY = 0;
    double calibZ = 0;

    for(size_t i = 0; i < calibSize;i++)

```

```

{
    readData();
    calibX += sensorVals.xf;
    calibY += sensorVals.yf;
    calibZ += sensorVals.zf;
}
offsetVals_.xf = static_cast<double>(calibX / ...
    static_cast<double>(calibSize));
offsetVals_.yf = static_cast<double>(calibY / ...
    static_cast<double>(calibSize));
offsetVals_.zf = static_cast<double>(calibZ / ...
    static_cast<double>(calibSize));
}

int AccelIMU::writeSingleReg()
{
    return 0;
}
/*
void AccelIMU::populateProtobuf(dronePosVec::dronePosition& dp)
{
    dp.clear_position();
    //dp.mutable_positiondot()->Add(sensorVals.xf); //alt method if ...
        needed, add_xxx should work fine however
    dp.add_position(sensorVals.xf);
    dp.add_position(sensorVals.yf);
    dp.add_position(sensorVals.zf);
}
*/

```

#### G.7.2.4 imuMainTest.cpp

```
#include "droneIMU.h"
#include <iostream>
#include <csignal>
#include <fstream>
#include <cstring>

volatile bool mainExit = false;
void sig_handler(int signum);
volatile bool running = true;

int main()
{
    int spiHandle = spiInit();
    if (spiHandle < 0)
    {
        std::cout<<"\n pigpio failed to initialize"<<std::endl;
        return -1;
    }

    GyroIMU gyro(spiHandle);
    AccelIMU accel(spiHandle);

    outputVals accelprev;
    outputVals gyroprev;
    outputVals accelint;
    outputVals gyroint;

    outputVals acceldubint;

    bool firstrun = false;

    std::ofstream csvWrite;
    csvWrite.open("numint.csv");
    csvWrite << "time,accelX,accelY,accelZ,gyroX,gyroY,gyroZ"<<std::endl;
    signal(SIGINT, &sig_handler);
    std::cout<<"starting read"<<std::endl;
    while(running) //TODO: make some way of exiting safely
    {
        if (firstrun)
        {
            accelprev = accel.sensorVals;
            gyroprev = gyro.sensorVals;
        }
        gyro.readData();
        accel.readData();
        if (firstrun)
        {
            accel.numIntergration(accelprev,accelint);
            gyro.numIntergration(gyroprev,gyroint);
            //accel.dubnumIntergration(accelprev,accelint,acceldubint);
            csvWrite << gyro.sensorVals.t.count() << ","<< ...
                accel.sensorVals.xf << ","<< accel.sensorVals.yf << ","<< ...
                accel.sensorVals.zf << ","<< gyro.sensorVals.xf << ","<< ...
                gyro.sensorVals.yf << ","<< gyro.sensorVals.zf<<std::endl;
            //csvWrite << gyro.sensorVals.t.count() << ","<< acceldubint.xf ...
            //      << ","<< acceldubint.yf << ","<< acceldubint.zf << ","<< ...
            //      gyroint.xf << ","<< gyroint.yf << ","<< gyroint.zf<<std::endl;
        }
    }
}
```

```

        std::this_thread::sleep_for(std::chrono::microseconds(10));
        firstrun = true;
        if (mainExit)
        {
            running = false;
        }
    }

    csvWrite.close();
    gpioEnd(spiHandle); //needs to be called before end of program
    return 0;
}

void sig_handler(int signum)
{
    if (signum == SIGINT)
    {
        mainExit = true;
        std::cerr << "\nInterrupt signal (" << signum << ") received.\n";
    }
    else
    {
        //restore default signal handler
        signal(signum, SIG_DFL);
    }
}

```

# Bibliography

- [1] Injisoft AB. *ASCII Table*. URL: <https://www.ascii-code.com/>. (Accessed: 23.05.2024).
- [2] Qualisys AB. *Understanding the calibration results*. URL: [https://docs.qualisys.com/getting-started/content/getting\\_started/running\\_your\\_qualisys\\_system/calibrating\\_your\\_system/understanding\\_the\\_calibration.htm](https://docs.qualisys.com/getting-started/content/getting_started/running_your_qualisys_system/calibrating_your_system/understanding_the_calibration.htm). (Accessed: 09.05.2024).
- [3] Fastbit Embedded Brain Academy. *I2C pull up resistor calculation, bus capacitance, and rise time lecture 9*. URL: <https://www.youtube.com/watch?v=CEPFWAPDy7o>. (Accessed: 21.05.2024).
- [4] National Aeronautics and Space Administration. *Downwash Effects Lift and Drag*. URL: <https://www.grc.nasa.gov/www/k-12/airplane/kitedown.html>. (Accessed: 22.05.2024).
- [5] National Aeronautics and Space Administration. *Shape Effects on Drag*. URL: <https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/shape-effects-on-drag/>. (Accessed: 22.05.2024).
- [6] AMS. *AS5600 12-Bit Programmable Contactless Potentiometer*. URL: <https://files.seeedstudio.com/wiki/Grove-12-bit-Magnetic-Rotary-Position-Sensor-AS5600/res/Magnetic%20Rotary%20Position%20Sensor%20AS5600%20Datasheet.pdf>. (Accessed: 16.5.2024).
- [7] H. Anton and A. Rorres C. Kaul. *Elementary Linear Algebra Applications Version*. Twelfth Edition. Hoboken: John Wiley & Sons, Inc, 2019. ISBN: 978-1-119-40672-3.
- [8] Atmel. *8 bit Atmel Microcontroller with 64 128Kbytes of ISP Flash and USB Controller*. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/doc7593.pdf>. (Accessed: 23.05.2024).
- [9] Atmel. *ATmega16U4/ATmega32U4*. URL: [https://ww1.microchip.com/downloads/en/devicedoc/atmel-7766-8-bit-avr-atmega16u4-32u4\\_datasheet.pdf](https://ww1.microchip.com/downloads/en/devicedoc/atmel-7766-8-bit-avr-atmega16u4-32u4_datasheet.pdf). (Accessed: 22.05.2024).
- [10] SURF B.V. *eduVPN*. URL: [https://play.google.com/store/apps/details?id=nl.eduvpn.app&hl=en\\_US&pli=1](https://play.google.com/store/apps/details?id=nl.eduvpn.app&hl=en_US&pli=1). (Accessed: 15.02.2024).
- [11] Barnert. *Sockets and multiprocessing*. URL: <http://stupidpythonideas.blogspot.com/2014/09/sockets-and-multiprocessing.html>. (Accessed: 19.05.2024).
- [12] J.M. Benum et al. “Modulbasert drone tilpasset autonom bruk i Motion Capture system.” Unpublished paper by University of Agder. 2018.
- [13] P. Bideau and E. Learned-Miller. *It’s Moving! A Probabilistic Model for Causal Motion Segmentation in Moving Camera Videos*. URL: [https://www.researchgate.net/publication/301879715\\_It's\\_Moving\\_A\\_Probabilistic\\_Model\\_for\\_Causal\\_Motion\\_Segmentation\\_in\\_Moving\\_Camera\\_Videos](https://www.researchgate.net/publication/301879715_It's_Moving_A_Probabilistic_Model_for_Causal_Motion_Segmentation_in_Moving_Camera_Videos). (Accessed: 25.01.2024).
- [14] J. Borge, M.S. Wad, and M. Økter. “Design and Development of a Drone Reinforcement Learning Arena.” Unpublished paper by University of Agder. 2022.
- [15] BOSCH. *BMI088 6-axis Motion Tracking for High-performance Applications*. URL: <https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmi088-ds001.pdf>. (Accessed: 27.04.2024).
- [16] P. Bristeau, E. Martin P. Salaün, and N. Petit. *The role of propeller aerodynamics in the model of a quadrotor UAV*. DOI: [10.23919/ECC.2009.7074482](https://doi.org/10.23919/ECC.2009.7074482). URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7074482>. (Accessed: 22.05.2024).
- [17] C. Bugs. *Xming for Windows*. URL: <https://xming.en.softonic.com/>. (Accessed: 15.02.2024).

- [18] D. Camera. *LUFA* (2013). URL: <http://www.fourwalledcubicle.com/LUFA.php>. (Accessed: 09.02.2024).
- [19] B. cherblanc. *Gemfan T 3025*. URL: <https://grabcad.com/library/gemfan-t-3025-1>. (Accessed: 22.05.2024).
- [20] The kernel development community. *Packed YUV formats*. URL: <https://docs.kernel.org/userspace-api/media/v4l/pixfmt-packed-yuv.html#:~:text=These%20formats%20commonly%20referred%20to,%2Dbit%20little%2DEndian%20words..> (Accessed: 25.01.2024).
- [21] RS Components. *RS PRO 2mtr USB 3.1 Type C Male to USB 3.0 Type A Male Cable*. URL: <https://docs.rs-online.com/6137/0900766b816e8347.pdf>. (Accessed: 12.02.2024).
- [22] ITP Physical Computing. *Accelerometers, Gyros, and IMUs: The Basics*. URL: <https://itp.nyu.edu/physcomp/lessons/accelerometers-gyros-and-imus-the-basics/>. (Accessed: 22.05.2024).
- [23] Intel Corporation. *Compare depth cameras*. URL: <https://www.intelrealsense.com/compare-depth-cameras/>. (Accessed: 01.03.2024).
- [24] Intel Corporation. *Depth Camera D435*. URL: <https://www.intelrealsense.com/depth-camera-d435/>. (Accessed: 08.01.2024).
- [25] Intel Corporation. *Intel® RealSense™ SDK 2.0 (v2.54.2)*. URL: <https://github.com/IntelRealSense/librealsense/releases/tag/v2.54.2>. (Accessed: 01.03.2024).
- [26] Intel Corporation. *Linux/Ubuntu - RealSense SDK 2.0 Build Guide*. URL: <https://dev.intelrealsense.com/docs/compiling-librealsense-for-linux-ubuntu-guide>. (Accessed: 12.02.2024).
- [27] Intel Corporation. *Projection in Intel RealSense SDK 2.0*. URL: <https://dev.intelrealsense.com/docs/projection-in-intel-realsense-sdk-20>. (Accessed: 12.02.2024).
- [28] Oracle Corporation. *What Are Sockets?* URL: <https://docs.oracle.com/cd/E19620-01/805-4041/6j3r8iu2k/index.html>. (Accessed: 20.05.2024).
- [29] Oracle Corporation. *What Is a Datagram?* URL: <https://docs.oracle.com/javase/tutorial/networking/datagrams/definition.html>. (Accessed: 20.05.2024).
- [30] cppreference. *Lambda expressions (since C++11)*. URL: <https://en.cppreference.com/w/cpp/language/lambda>. (Accessed: 20.05.2024).
- [31] cppreference. *std::queue*. URL: <https://en.cppreference.com/w/cpp/container/queue>. (Accessed: 16.05.2024).
- [32] Analog Devices. *Accelerometer and Gyroscopes Sensors: Operation, Sensing, and Applications*. URL: <https://www.analog.com/en/resources/technical-articles/accelerometer-and-gyroscopes-sensors-operation-sensing-and-applications.html>. (Accessed: 22.05.2024).
- [33] P. Dhaker. *Introduction to SPI Interface*. URL: <https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html>. (Accessed: 21.05.2024).
- [34] N. DiFilippo. *Realsense [realsense.py]*. <https://github.com/nickredsox/youtube/blob/master/Robotics/realsense.py>. 2023.
- [35] Dychen. *A Comprehensive Guide to Resettable Fuses*. URL: <https://community.element14.com/technologies/experts/b/comprehensive-guides/posts/a-comprehensive-guide-to-resettable-fuses>. (Accessed: 22.05.2024).
- [36] Hewlett Packard Enterprise. *CIDR Conversion Table*. URL: <https://techlibrary.hpe.com/docs/otlink-wo/CIDR-Conversion-Table.html>. (Accessed: 21.05.2024).
- [37] J.A. Farrell. *AIDED NAVIGATION GPS with High Rate Sensors*. eng. The McGraw-Hill, 2008. ISBN: 0-07-164266-8.

- [38] M. Fiala. “ARTag, a fiducial marker system using digital techniques.” In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 2. 2005, 590–596 vol. 2. DOI: [10.1109/CVPR.2005.74](https://doi.org/10.1109/CVPR.2005.74).
- [39] N. Fiedler. *Distributed Multi Object Tracking with Direct FCNN Inclusion in RoboCup Humanoid Soccer*. URL: [https://www.researchgate.net/publication/336614405\\_Distributed\\_Multi\\_Object\\_Tracking\\_with\\_Direct\\_FCNN\\_Inclusion\\_in\\_RoboCup\\_Humanoid\\_Soccer](https://www.researchgate.net/publication/336614405_Distributed_Multi_Object_Tracking_with_Direct_FCNN_Inclusion_in_RoboCup_Humanoid_Soccer). (Accessed: 25.01.2024).
- [40] D. Florian. *Stepper Motors*. URL: <https://www.drdflo.com/pages/Guides/How-to-Build-a-3D-Printer/Stepper-Motor.html>. (Accessed: 22.05.2024).
- [41] W. Förstner and E. Gülch. “A fast operator for detection and precise location of distinct points, corners and centres of circular features.” In: *Proc. ISPRS intercommission conference on fast processing of photogrammetric data*. Vol. 6. Interlaken. 1987, pp. 281–305.
- [42] Inc Fortinet. *What is Transmission Control Protocol TCP/IP?* URL: <https://www.fortinet.com/resources/cyberglossary/tcp-ip>. (Accessed: 20.05.2024).
- [43] Python Software Foundation. *multiprocessing — Process-based parallelism*. URL: <https://docs.python.org/3/library/multiprocessing.html>. (Accessed: 20.05.2024).
- [44] S. Garrido-Jurado et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion.” In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: <https://doi.org/10.1016/j.patcog.2014.01.005>. URL: <https://www.sciencedirect.com/science/article/pii/S0031320314000235>.
- [45] GeeksForGeeks. *What is Network Port?* URL: <https://www.geeksforgeeks.org/what-is-network-port/>. (Accessed: 20.05.2024).
- [46] D. Gray. *Distortion 101 - Lens vs. Perspective*. URL: <https://www.drewgrayphoto.com/learn/distortion101>. (Accessed: 21.02.2024).
- [47] THE SCIENCE MUSEUM GROUP. *INTRODUCTION TO THE CAMERA OBSCURA*. URL: <https://blog.scienceandmediamuseum.org.uk/introduction-camera-obscura/>. (Accessed: 13.02.2024).
- [48] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. eng. 2nd Edition. Cambridge: Cambridge University Press, 2004. ISBN: 9780521540513.
- [49] J.L. Haugenes and T.H. Evensen. “3DOF Quadcopter Platform.” Unpublished paper by University of Agder. 2017.
- [50] E. Helle et al. “Flight Controller Design and Implementation for UiA Drone Project.” Unpublished paper by University of Agder. 2022.
- [51] G. Hollows and N. James. *Distortion*. URL: <https://www.edmundoptics.com/knowledge-center/application-notes/imaging/distortion/#:~:text=How%20is%20Distortion%20Specified%3F,measurement%20algorithms%20are%20not%20used..> (Accessed: 07.02.2024).
- [52] J. Hopkins. *What is a Register in Processors/CPUs?* URL: <https://www.totalphase.com/blog/2023/05/what-is-register-in-cpu-how-does-it-work/>. (Accessed: 23.05.2024).
- [53] The MathWorks Inc. *Calibration Patterns*. URL: <https://se.mathworks.com/help/vision/ug/calibration-patterns.html>. (Accessed: 23.02.2024).
- [54] Infineon. *BSC007N04LS6*. URL: [https://www.infineon.com/dgdl/Infineon-BSC007N04LS6-DataSheet-v02\\_03-EN.pdf?fileId=5546d462689a790c0168bcd9a63a4815](https://www.infineon.com/dgdl/Infineon-BSC007N04LS6-DataSheet-v02_03-EN.pdf?fileId=5546d462689a790c0168bcd9a63a4815). (Accessed: 21.05.2024).
- [55] IPC. *Generic Standard on Printed Board Design*. URL: [https://www-eng.lbl.gov/~shuman/NEXT/CURRENT\\_DESIGN/TP/MATERIALS/IPC-2221A\(L\).pdf](https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A(L).pdf). (Accessed: 20.02.2024).
- [56] A. Janota et al. *Improving the Precision and Speed of Euler Angles Computation from Low-Cost Rotation Sensor Data*. DOI: <https://doi.org/10.3390/s150307016>. URL: <https://www.mdpi.com/1424-8220/15/3/7016>. (Accessed: 22.03.2024).

- [57] P. Jin, P. Matikainen, and S.S. Srinivasa. *Sensor Fusion for Fiducial Tags: Highly Robust Pose Estimation from Single Frame RGBD*. URL: <https://personalrobotics.cs.washington.edu/publications/jin2017rgbdtags.pdf>. (Accessed: 08.04.2024).
- [58] JLCPCB. *PCB Manufacturing & Assembly Capabilities*. URL: <https://jlcpcb.com/capabilities pcb-capabilities>. (Accessed: 20.02.2024).
- [59] O. Kalachev. *ArUco markers generator!* URL: <https://chev.me/arucogen/>. (Accessed: 25.01.2024).
- [60] R. Keim. *Reverse Polarity Protection: How to Protect Your Circuits Using Only a Diode*. URL: <https://www.allaboutcircuits.com/technical-articles/how-to-protect-your-circuits-using-only-a-diode/>. (Accessed: 22.05.2024).
- [61] M. Kerrisk. *open(2) — Linux manual page*. URL: <https://man7.org/linux/man-pages/man2/open.2.html>. (Accessed: 21.05.2024).
- [62] C. Kirkham. *Dismantling ARToolkit Part 2: Contour Detection*. URL: <http://chriskirkham.co.uk/2011/06/27/dismantling-artoolkit-part-2-contour-detection/>. (Accessed: 25.01.2024).
- [63] F. Koyama. “Recent Advances of VCSEL Photonics.” In: 24.12 (2006), pp. 4502–4513. DOI: [10.1109/JLT.2006.886064](https://doi.org/10.1109/JLT.2006.886064).
- [64] K.H. Kruithof and M. Egeland. “State Estimator using Hybrid Kalman and Particle Filter for Indoor UAV Navigation.” Unpublished paper by University of Agder. 2021.
- [65] S. Kudrle et al. “Fingerprinting for Solving A/V Synchronization Issues within Broadcast Environments.” In: *SMPTE Motion Imaging Journal* 120.5 (2010), pp. 36–46. DOI: [10.5594/j18059XY](https://doi.org/10.5594/j18059XY).
- [66] O. Liang. *How To Use MOSFET – An Electronics Beginner’s Tutorial*. URL: <https://oscarliang.com/how-to-use-mosfet-beginner-tutorial/>. (Accessed: 23.05.2024).
- [67] Littelfuse. *2920L Series Surface Mount*. URL: [https://www.littelfuse.com/~/media/electronics/datasheets/resettable\\_ptcs/littelfuse\\_ptc\\_2920l\\_datasheet.pdf.pdf](https://www.littelfuse.com/~/media/electronics/datasheets/resettable_ptcs/littelfuse_ptc_2920l_datasheet.pdf.pdf). (Accessed: 16.05.2024).
- [68] Inc. Littelfuse. *RXF300*. URL: [https://www.littelfuse.com/~/media/electronics/product\\_specifications/resettable\\_ptcs/littelfuse\\_ptc\\_rxef300\\_product\\_specification.pdf.pdf](https://www.littelfuse.com/~/media/electronics/product_specifications/resettable_ptcs/littelfuse_ptc_rxef300_product_specification.pdf.pdf). (Accessed: 21.05.2024).
- [69] Greater Zurich Area Ltd. *Innovation Park Zurich gains drone testing arena*. URL: <https://www.greaterzuricharea.com/en/news/innovation-park-zurich-gains-drone-testing-area>. (Accessed: 21.03.2024).
- [70] S. Mallick. *Why does OpenCV use BGR color format ?* URL: <https://learnopencv.com/why-does-opencv-use-bgr-color-format/>. (Accessed: 14.02.2024).
- [71] S. Mallick and K. Sadekar. *Camera Calibration using OpenCV*. URL: <https://learnopencv.com/camera-calibration-using-opencv/>. (Accessed: 23.02.2024).
- [72] MatWeb. *Elektro-Isola G-Etronax EP FR4 Epoxy, Glass Fabric Reinforcement, Light Yellow, Sheets*. URL: <https://www.matweb.com/search/DataSheet.aspx?MatGUID=3c5f252f235844fb9ffec6d1>. (Accessed: 23.05.2024).
- [73] MatWeb. *Hexcel® UHM Carbon Fiber (3000 Filaments)*. URL: <https://www.matweb.com/search/DataSheet.aspx?MatGUID=ab6c2c8416e5425fb83b032877c22ea47>. (Accessed: 23.05.2024).
- [74] METCALC. *Round head screw calculator*. URL: <https://en.metcalc.info/calc-fasteners/screw/>. (Accessed: 19.05.2024).
- [75] N. Nielsen. *Learn Camera Calibration in Python with OpenCV: Complete Step-by-Step Guide with Python Script*. URL: <https://www.youtube.com/watch?v=3h7wgR5fYik>. (Accessed: 29.01.2024).

- [76] N.S. Nise. *Nise's Control Systems Engineering*. Global Edition. John Wiley & sons, 2015. ISBN: 978-1-119-38297-3.
- [77] R.L Norton. *Machine Design, an Integrated Approach*. eng. 6th Edition. Pearson, 2019. ISBN: 0-13-518423-1.
- [78] S.O. Nyberg. *Statistikk - en bayesiansk tilnærming*. 2. opplag. Oslo: Universitetsforlaget, 2017. ISBN: 978-82-15-02637-4.
- [79] OMNIVISION. *1/6" CMOS 1080p (1920 x 1080) HD Image Sensor with PureCel Technology*. URL: <https://www.ovt.com/products/ov2740/>. (Accessed: 31.01.2024).
- [80] OMNIVISION. *B&W CMOS 1 Megapixel (1280x800) Image Sensor with OmniPixel 3-GS Technology*. URL: <https://www.ovt.com/products/ov09282-ga4a/>. (Accessed: 13.02.2024).
- [81] OpenCV. *About*. URL: <https://opencv.org/about/>. (Accessed: 02.02.2024).
- [82] OpenCV. *Camera Calibration*. URL: [https://docs.opencv.org/4.x/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html). (Accessed: 08.02.2024).
- [83] OpenCV. *Camera Calibration and 3D Reconstruction*. URL: [https://docs.opencv.org/4.x/d9/d0c/group\\_\\_calib3d.html](https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html). (Accessed: 21.02.2024).
- [84] OpenCV. *ColorMaps in OpenCV*. URL: [https://docs.opencv.org/4.x/d3/d50/group\\_\\_imgproc\\_\\_colormap.html](https://docs.opencv.org/4.x/d3/d50/group__imgproc__colormap.html). (Accessed: 14.02.2024).
- [85] OpenCV. *Detection of ArUco Markers*. URL: [https://docs.opencv.org/4.x/d5/dae/tutorial\\_aruco\\_detection.html](https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html). (Accessed: 25.01.2024).
- [86] OpenCV. *Detection of ChArUco Boards*. URL: [https://docs.opencv.org/3.4/df/d4a/tutorial\\_charuco\\_detection.html](https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html). (Accessed: 23.02.2024).
- [87] OpenCV. *OpenCV modules*. URL: <https://docs.opencv.org/4.6.0/>. (Accessed: 23.02.2024).
- [88] OpenCV. *Perspective-n-Point (PnP) pose computation*. URL: [https://docs.opencv.org/4.x/d5/d1f/calib3d\\_solvePnP.html](https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html). (Accessed: 08.04.2024).
- [89] I. Pandzic. *Clock Synchronization and Monotonic Clocks*. URL: <https://inelpandzic.com/articles/clock-synchronization-and-monotonic-clocks/>. (Accessed: 20.05.2024).
- [90] V.M.N. Passaro et al. "Gyroscope Technology and Applications: A Review in the Industrial Perspective." In: *Sensors (Basel)* (2017). DOI: <https://doi.org/10.3390%2Fs17102284>. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5677445/>.
- [91] ERUDIRE PLUS. *CFD Analysis of a Quad Rotor (UAV) with SolidWorks 2020*. URL: <https://www.youtube.com/watch?v=P2NJFXZw8Ng>. (Accessed: 7.03.2024).
- [92] Pololu. *DRV8825 Stepper Motor Driver Carrier, High Current*. URL: <https://www.pololu.com/product/2133>. (Accessed: 15.02.2024).
- [93] LEARNING POWER. *3 BLADE PROPELLER PARAMETRIC ANALYSIS/FLOW SIMULATION IN SOLIDWORKS*. URL: [https://www.youtube.com/watch?v=QKy4mi\\_UoTA](https://www.youtube.com/watch?v=QKy4mi_UoTA). (Accessed: 29.02.2024).
- [94] PPprint. *P-filament 721 natural*. URL: <https://www.ppprint.de/en/produkt/filament/>. (Accessed: 22.05.2024).
- [95] PPprint. *Technisches Datenblatt: P-filament*. URL: <https://www.ppprint.de/wp-content/uploads/2021/01/P-filament-721-Datenblatt-deutsch.pdf>. (Accessed: 23.05.2024).
- [96] PrimaCreator. *PrimaSelect™ PLA Technical Data*. URL: <https://primacreator.com/products/primaselect%E2%84%A2-pla?variant=61738769867>. (Accessed: 23.05.2024).
- [97] Rust Project. *Module rppal::spi*. URL: <https://docs.rs/rppal/latest/rppal/spi/index.html>. (Accessed: 03.05.2024).
- [98] PyImageSearch. *Detecting ArUco markers with OpenCV and Python*. URL: <https://pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/>. (Accessed: 29.01.2024).

- [99] M. Rehak. *Integrated Sensor Orientation on Micro Aerial Vehicles*. URL: [https://www.researchgate.net/publication/314259443\\_Integrated\\_Sensor\\_Orientation\\_on\\_Micro\\_Aerial\\_Vehicles](https://www.researchgate.net/publication/314259443_Integrated_Sensor_Orientation_on_Micro_Aerial_Vehicles). (Accessed: 25.01.2024).
- [100] RLS. *AksIM-4™ Big Rings*. URL: <https://www.rls.si/eng/aksim-4-rotary-absolute-magnetic-encoder>. (Accessed: 20.05.2024).
- [101] Open Robotics. *Ubuntu (Debian packages)*. URL: <https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html>. (Accessed: 13.02.2024).
- [102] D. Rose. *Rotations in Three-Dimensions: Euler Angles and Rotation Matrices*. URL: [https://danceswithcode.net/engineeringnotes/rotations\\_in\\_3d/rotations\\_in\\_3d\\_part1.html](https://danceswithcode.net/engineeringnotes/rotations_in_3d/rotations_in_3d_part1.html). (Accessed: 23.03.2024).
- [103] G. Sanderson and B. Eater. *Visualizing quaternions An explorable video series*. URL: <https://eater.net/quaternions>. (Accessed: 08.03.2024).
- [104] w3 schools. *C++ Polymorphism*. URL: [https://www.w3schools.com/cpp/cpp\\_polymorphism.asp](https://www.w3schools.com/cpp/cpp_polymorphism.asp). (Accessed: 16.05.2024).
- [105] NASA Science. *Visible Light*. URL: [https://science.nasa.gov/ems/09\\_visiblelight](https://science.nasa.gov/ems/09_visiblelight). (Accessed: 15.02.2024).
- [106] R. Sheldon. *Data abstraction*. URL: <https://www.techtarget.com/whatis/definition/data-abstraction>. (Accessed: 20.05.2024).
- [107] Cadence PCB Solutions. *Learn How to Limit Current to LED*. URL: <https://resourcespcb.cadence.com/blog/2022-learn-how-to-limit-current-to-led>. (Accessed: 22.05.2024).
- [108] Y. Song and D. Scaramuzza. *Reinforcement Learning for Agile Flight: From Perception to Action*. URL: <https://ippc-iros23.github.io/papers/song.pdf>. (Accessed: 01.02.2024).
- [109] J. Steward. *Camera Modeling: Exploring Distortion and Distortion Models, Part II*. URL: <https://www.tangramvision.com/blog/camera-modeling-exploring-distortion-and-distortion-models-part-ii>. (Accessed: 21.02.2024).
- [110] B. Stroustrup. “What is object-oriented programming?” In: *IEEE Software* 5.3 (1988), pp. 10–20. DOI: [10.1109/52.2020](https://doi.org/10.1109/52.2020).
- [111] E. Taggart and M. Cole. *The History of Camera Obscura and How It Was Used as a Tool To Create Art in Perfect Perspective*. URL: <https://mymodernmet.com/camera-obscura/>. (Accessed: 13.02.2024).
- [112] S. Tatham. *mksvg.py*. URL: <https://git.tartarus.org/?p=simon/putty.git;a=blob;f=icons/mksvg.py;hb=HEAD>. (Accessed: 15.02.2024).
- [113] J. Taylor. *Understanding Host Name Resolution*. URL: <https://newsletters.inficon.com/FabGuard/May2018/UnderstandingHostNameResolution.html>. (Accessed: 20.05.2024).
- [114] CPPreference Team. *std::this\_thread::sleep\_for*. URL: [https://en.cppreference.com/w/cpp/thread/sleep\\_for](https://en.cppreference.com/w/cpp/thread/sleep_for). (Accessed: 20.05.2024).
- [115] CPPreference Team. *std::unique\_ptr*. URL: [https://en.cppreference.com/w/cpp/memory/unique\\_ptr](https://en.cppreference.com/w/cpp/memory/unique_ptr). (Accessed: 20.05.2024).
- [116] Intel Realsense Team. *pyrealsense2.pipeline*. URL: [https://intelrealsense.github.io/librealsense/python\\_docs/\\_generated/pyrealsense2.pipeline.html](https://intelrealsense.github.io/librealsense/python_docs/_generated/pyrealsense2.pipeline.html). (Accessed: 07.02.2024).
- [117] Protocol Buffer Team. *Language Guide (proto 3)*. URL: <https://protobuf.dev/programming-guides/proto3/>. (Accessed: 22.05.2024).
- [118] Protocol Buffer Team. *Overview*. URL: <https://protobuf.dev/overview/>. (Accessed: 20.05.2024).
- [119] Sierra Circuits Team. *What is the Use of a Decoupling Capacitor?* URL: <https://www.protoexpress.com/blog/decoupling-capacitor-use/>. (Accessed: 22.05.2024).

- [120] Intel technologies. *Intel® RealSense™ D400 Series Dynamic Calibration Tool*. URL: <https://www.intel.com/content/www/us/en/download/645988/intel-realsense-d400-series-dynamic-calibration-tool.html>. (Accessed: 07.02.2024).
- [121] Inc. The MathWorks. *Evaluating Goodness of Fit*. URL: [https://se.mathworks.com/help/curvefit/evaluating-goodness-of-fit.html#bq\\_5kwr-3](https://se.mathworks.com/help/curvefit/evaluating-goodness-of-fit.html#bq_5kwr-3). (Accessed: 12.05.2024).
- [122] Inc. The MathWorks. *Understanding Kalman Filters*. URL: <https://se.mathworks.com/videos/series/understanding-kalman-filters.html>. (Accessed: 02.05.2024).
- [123] ThomaL2002. *RealSense D435 Distortion Model*. URL: <https://support.intelrealsense.com/hc/en-us/community/posts/26224885823763-RealSense-D435-Distortion-Model>. (Accessed: 12.01.2024).
- [124] Toshiba. *MOSFET Gate Drive Circuit*. URL: [https://toshiba.semicon-storage.com/info/application\\_note\\_en\\_20180726\\_AKX00068.pdf?did=59460](https://toshiba.semicon-storage.com/info/application_note_en_20180726_AKX00068.pdf?did=59460). (Accessed: 20.02.2024).
- [125] Toshiba. *Resistors are often inserted between a CPU and MOSFETs. Why are these resistors necessary?* URL: <https://toshiba.semicon-storage.com/ap-en/semiconductor/knowledge/faq/mosfet/resistors-are-often-inserted-between-a-cpu-and-mosfetswhy-are-th.html>. (Accessed: 20.02.2024).
- [126] Toshiba. *TB6612FNG*. URL: <https://www.pololu.com/file/0J86/TB6612FNG.pdf>. (Accessed: 15.02.2024).
- [127] M. Ulrich, C. Steger, and C. Wiedemann. *Machine Vision Algorithms and Applications*. Second Edition. München: Wiley-VCH, 2018. ISBN: 978-3-527-41365-2.
- [128] J. Valdez. *How to calculate bus capacitance*. URL: <https://e2e.ti.com/support/interface-group/interface/f/interface-forum/470156/how-to-calculate-bus-capacitance>. (Accessed: 21.05.2024).
- [129] Vertex42. *How to Create a Gantt Chart in Google Sheets*. URL: <https://www.youtube.com/watch?v=8eKk0M2zGIk>. (Accessed: 19.01.2024).
- [130] M. Voss, R. Asenjo, and J. Reinders. *Pro TBB*. eng. 1st Edition. Apress Berkeley, 2019. ISBN: 978-1-4842-4397-8.
- [131] w3schools. *JSON - Introduction*. URL: [https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp). (Accessed: 20.05.2024).
- [132] E.W. Weisstein. *Rotation Matrix*. URL: <https://mathworld.wolfram.com/RotationMatrix.html>. (Accessed: 22.03.2024).
- [133] Z. Zhang. *A Flexible New Technique for Camera Calibration*. URL: <https://www.microsoft.com/en-us/research/publication/a-flexible-new-technique-for-camera-calibration/?from=https://research.microsoft.com/en-us/people/zhang/Papers/TR98-71.pdf&type=exact>. (Accessed: 23.02.2024).