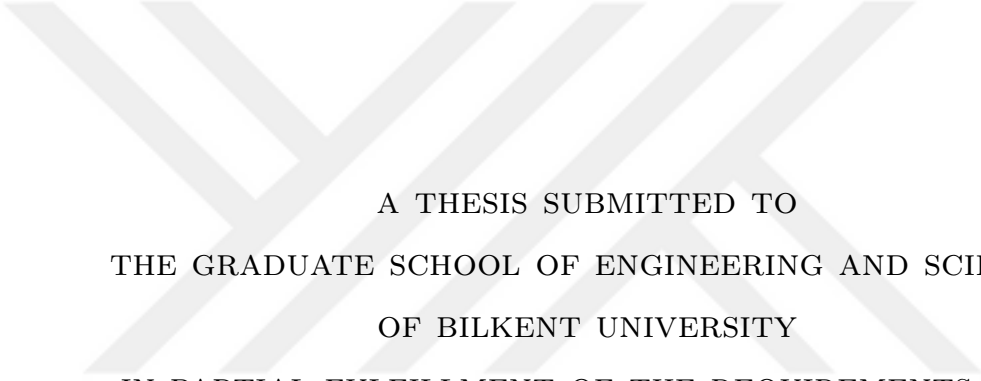


A MINIATURE, FOLDABLE, COLLISION RESILIENT QUADCOPTER



A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

By
Alihan Bakır
June 2023

A Miniature, Foldable, Collision Resilient Quadcopter

By Alihan Bakır

June 2023

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Onur Özcan(Advisor)

Yıldıray Yıldız

Ali Emre Turgut

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

A MINIATURE, FOLDABLE, COLLISION RESILIENT QUADCOPTER

Alihan Bakır

M.S. in Mechanical Engineering

Advisor: Onur Özcan

June 2023

In the fields of surveillance, mapping and security, the use of unmanned aerial vehicles (UAVs), is becoming inevitable day by day, especially with their autonomous movement capabilities. The main reason for the increasing use of unmanned aerial vehicles is their ability to map and survey unknown and dangerous places such as caves without risking human life. At the same time, the ability to conduct aerial surveillance in full autonomy for public safety and health is also an important factor. Today, UAVs are used for many purposes, such as cave mapping and surveying, detection and intervention of forest fires, and inspection of outdoor and crowded areas for security purposes, often accompanied by an operator.

Quadcopters are also used for missions that do not require long duration flight, as they are able to both hold position and ensure a highly stable flight. In addition to their many advantages, these UAVs are very sensitive to even the slightest impact, so their fully autonomous flight is still limited to controlled areas. Various studies are being conducted to increase the crash resistance of the quadcopters. Among these researches, there are different ideas such as protective shells and bumpers that surround the UAV and absorb the impact in case of collisions. The spherical cases that surround the UAV are usually in a mesh structure to be lightweight and not obstruct the airflow. Bumpers designed to protect the most sensitive parts of the UAV, such as the motors and propellers, are insufficient to protect the body. For these reasons, making different parts of the UAV from more flexible materials will eliminate the vulnerability of the UAV and increase its resistance to collisions.

In this thesis, in order to increase the impact resistance of quadcopters and to ensure that they do not break, robots with some flexible and some rigid parts

were tested and the results of these tests were evaluated in detail. During these tests, the effects of the compliance of the robot's arms and the compliance of the bumpers protecting the propellers upon the impact were analyzed. In order to make this comparison, flexible and rigid robot bodies with dimensions as close as possible to each other, as well as rigid and flexible bumpers of similar size and structure were designed. The flexible bumper and body were produced by cutting PET sheets via a laser cutter and folding them in an origami-inspired pattern. This production method adds flexibility thanks to the thinness of the PET sheets and structural rigidity thanks to the origami-inspired folding technique. To increase the flexibility of the robot in the event of a collision and the stability of the motors, inserts made of TPU are inserted into the body.

In addition to impact resilience, this thesis also discusses a soft sensor that can be attached to collision-resilient drones. This sensor, made of conductive TPU, allows the robot to sense its surroundings by giving it a sense of touch. Thanks to the flexible sensor, robots can detect when a collision occurs and react accordingly. This sensor works entirely based on the flexibility of the UAV's bumpers and senses the bending of these bumpers. Therefore, such a sensor cannot be used on rigid hulls and bumpers as they do not bend.

Keywords: Drone, Quadcopter, Control, PID, Collision Resilience, Compliance, Soft Sensor, Sensing Impacts, Autonomy, Sensorized Quadcopter, Sense of Touch.

ÖZET

MINYATÜR, KATLANABİLİR, ÇARPŞMAYA DİRENÇLİ QUADCOPTER

Alihan Bakır
Makine Mühendisliği, Yüksek Lisans
Tez Danışmanı: Onur Özcan
Haziran 2023

Gözlem, haritalama ve güvenlik alanlarında, özellikle otonom hareket kabiliyetleri ve insansız hava araçlarının kullanımı gün geçtikçe kaçınılmaz hale gelmektedir. İnsansız hava araçlarının kullanımının artışıdaki temel sebep, insan hayatını riske atmadan mağara gibi bilinmeyen ve tehlikeli yerlerde haritalandırma ve araştırma yapabilme yeteneğine sahip olmalarıdır. Aynı zamanda da toplum güvenliği ve sağlığı için tam otonom bir şekilde havadan gözlem yapabilmeleri de önemli bir faktördür. İnsansız hava araçları günümüzde, çoğunlukla bir operatör eşliğinde mağara haritalama ve inceleme, orman yangınlarının tespiti ve müdahalesi, açık ve kalabalık alanların güvenlik amacıyla incelenmesi gibi birçok amaçla kullanılmaktadır.

Uzun süreli uçuş gerektirmeyen görevlerde de hem pozisyon koruma becerisi hem de oldukça stabil uçuş sağlayabildiğinden dört adet döner kanat barındıran insansız hava araçları kullanılmaktadır. Bu tip İHA'lar birçok artışının yanında en ufak darbeye karşı bile çok hassas olduklarından tam otonom uçuşları henüz kontrollü bölgelerle sınırlıdır. Bu döner kanatlı İHA'ların çarpışma direncini arttırmak için çeşitli çalışmalar yapılmaktadır. Bu araştırmalar arasında, İHA'nın etrafını saran ve çarpmalarda darbeyi emen koruyucu kabuklar, tamponlar gibi farklı fikirler bulunmaktadır. İHA'nın etrafını sarmalayan küresel kılıflar genelde hafif olması ve hava akışını engellememesi için ağ yapısında olurlar. İHA'nın en hassas parçaları olan motor ve pervaneleri korumak için tasarlanmış olan tamponlar ise gövdeyi korumakta yetersiz kalmaktadırlar. Bu sebeplerden dolayı, İHA'nın farklı parçalarının daha esnek malzemelerden yapılması İHA'nın kırılma direncini düşürürken, çarpmalara karşı direncini arttıracaktır.

Bu tez kapsamında, İHA'ların çarpma direncini arttırmak ve kırılmamalarını sağlamak amacıyla bazı parçaları esnek bazıları sert tasarlanmış robotlar test

edilmiş ve bu testlerin sonuçları ayrıntılı olarak değerlendirilmiştir. Bu testler sırasında robotun kollarının esnekliğinin ve pervaneleri koruyan tamponların esnekliğinin çarpışmaya olan etkisi incelenmiştir. Bu karşılaştırmayı yapabilmek için, birbirine mümkün olduğunca yakın boyutlarda esnek ve sert robot gövdeleriyle birlikte, benzer boyut ve yapıda sert ve esnek tamponlar tasarlanmıştır. Esnek tampon ve gövde üretimi PET kağıtların lazer kesicide kesilip origamiden esinlenilerek katlanmasıyla üretilmiştir. Bu üretim metodu, PET kağıtların inceliği sayesinde esneklik katarken, origamiden tekniği sayesinde de yapısal sağlamlık katar. Robotun çarpışma anında esnekliğini ve motorların stabilitesini arttırmak için gövdeye TPU'dan üretilen çubuklar yerleştirilmiştir.

Çarpışma dayanıklılığına ek olarak, bu tezde, çarpışmaya dayanıklı İHA'lara takılabilecek bir esnek sensörden de bahsedilmiştir. İletken TPU'dan üretilen bu sensor robota dokunma hissi vererek etrafını algılayabilmesini sağlamaktadır. Üzerindeki esnek sensör sayesinde robotlar bir çarpışma gerçekleştiğinde bunu algılayabilir ve buna uygun bir tepki verebilir. Bu sensör tamamıyla İHA'nın tamponlarının esnek oluşuna dayanarak çalışır ve bu tamponların esnemesini hisseder. Dolayısıyla sert gövde ve tamponlarda esneme olmadığından böyle bir sensör kullanılamaz.

Anahtar sözcükler: Drone, Quadcopter, Kontrol, PID, Çarpışma Dayanıklılığı, Esneklik, Esnek Sensör, Darbeleri Algılama, Otonomi, Sensörlü Quadcopter, Dokunma Hissi.

Acknowledgement

I would like to express my sincere gratitude and appreciation to everyone who contributed this thesis and helped me in any way, for their support and contributions throughout my journey in completing this master's thesis.

First and foremost, I extend my heartfelt thanks to my advisor, Dr. Onur Ozcan, for his invaluable guidance, mentorship, and unwavering support. Your expertise, patience, and commitment to excellence have played a crucial role in shaping this thesis and my academic development. I am profoundly grateful for the knowledge and wisdom you have shared with me. Without your insights regarding the life itself as well as the work we have done, I could not be where I am right now.

I would also like to thank Dr. Yildiray Yildiz, coadvisor of one of the projects I worked on. Your expertise and valuable input greatly influenced the outcomes of my research, and I am grateful for your guidance throughout the project.

I am fortunate to have had the opportunity to work alongside exceptional lab mates who have not only been great colleagues but also friends. Mustafa Ugur and Doga Ozbek, your companionship, discussions, and academic support have been instrumental in shaping my ideas and refining my work. Whether it was through late-night study sessions, brainstorming ideas, or simply being there to provide a listening ear, your presence has made this academic pursuit all the more meaningful and enjoyable. Your enthusiasm and dedication to our shared goals have made this journey both productive and enjoyable. And Amirali Abazari, your help and input throughout the projects that we have worked on as well as your friendship were irreplaceable.

I would also like to extend my sincere appreciation to my beloved friends, Kerem, Kaan, Talip, Koklem, and Secil. Your friendship has been a constant source of joy, laughter, and support throughout my master's journey. Your friendship also has been effective in maintaining my motivation and perspective during

challenging times. Thank you for everything you have done to help me, being an integral part of my life and for always standing by me.

I extend my heartfelt appreciation to my companion, Zeynep. Your unwavering support, understanding, and encouragement have been invaluable throughout this demanding process. Your belief in me and your presence by my side have provided the motivation and strength to overcome challenges and reach the finish line. I am grateful for your love and intimacy. I owe my sanity to your faith in me.

Last but certainly not least, I would like to express my deepest gratitude to my family, my beloved sister, father and mother. Your unconditional love, support, and encouragement have been the foundation of my success. Your sacrifices and belief in my abilities have constantly inspired me to strive for excellence. I am eternally grateful for your persistent support and for always being there for me, both during this thesis journey and throughout my academic career. I won't be able to achieve everything I have done without the help of my family, especially my sister, Ilke Bakir.

To all those mentioned above and to anyone else who has contributed in any way, I extend my sincerest thanks. This thesis would not have been possible without your assistance, guidance, and support.

I acknowledge the Scientific and Technological Research Council of Türkiye (TUBITAK), for the financial support they provided to this research by Grant number: 118E937.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Research Objectives	2
1.3	Structure of Thesis	4
2	Design	6
2.1	Design of the Robot	6
2.2	Sensor Design	10
3	Electronics	13
3.1	Component Selection	13
3.2	PCB Design	14
4	Control and Operation	17
4.1	Attitude Controller	18

4.2	High-Level Controllers	21
4.2.1	Yaw Angle Controller	22
4.2.2	Altitude Controller	23
4.2.3	Position Controller	28
5	Experiments and Results	33
5.1	Attitude Controller	33
5.2	High-Level Controllers	36
5.2.1	Altitude Controller	36
5.2.2	Yaw Angle Controller	41
5.2.3	Position Controller	42
5.3	Sensorized Collision Resilient Aerial Robot (SCoReR)	58
5.4	Effect of Compliance in Drone Body	61
6	Conclusion and Future Work	71
A	Data	77

List of Figures

2.1	<i>Sensorized Collision Resilient aerial Robot (SCoReR)</i>	8
2.2	Foldable arm design; the extension locks (1), the groove-and-tongue joint (2), and the folded and unfolded motor mounts (3) .	9
2.3	(a) Landing gear installation on the central housing and arms (b) Illustration of the foldable bumper installation on the arms. Four extensions are fastened on the last lock on the arm (left), the inclination of the bumpers help the drone recover about the pitching/rolling axis in collisions (right)	10
2.4	(a) Unfolded design of the new bumper design, (b) Folded and mounted version of the new bumper design	11
2.5	(a) CAD model of the soft sensor, (b) Picture of the soft sensor .	12
3.1	First version of the PCB used on the robot. (a) Top view, (b) Bottom view, (c) Schematics of the PCB	15
3.2	Two separate PCB designs with same connection diagram. (a) The one with sensor chips, (b) The one with sensors as modules, (c) Connection diagram of both PCBs	16

3.3	Final version of the PCB used on the robot. (a) Board view, (b) Connection diagram of the PCB	16
4.1	Simple PID controller used in the initial flight tests. (a) For roll and pitch angles, (b) For yaw angle	19
4.2	Cascaded PID loop schematics for roll and pitch angle control . .	20
4.3	Cascaded PID loop schematics for yaw angle control	23
4.4	1-D Kalman estimator for digital barometer data	25
4.5	(a) First altitude controller with no nominal throttle value, (b) The final version with nominal throttle value. Sensor data is from the digital barometer for outdoor flights and motion capture data for indoor flights	27
4.6	(a) When the integral term of the altitude controller is increased from 180 to 230, the UAV cannot take off. (b) When the K_i constant of the altitude controller is increased, the UAV can take off but cannot reach a stable point.	28
4.7	Schematics of GPS position controller	31
4.8	GPS data gathered for around 5 minutes while the sensor is stationary. (a) x vs y scatter plot (b) time vs x&y	32
5.1	Robot's roll and pitch angles vs. sample number. The angle values are multiplied by 100 and samples are gathered in 6 seconds. (a) For roll angle, (b) For pitch angle	34
5.2	Result of the altitude flight with PID constants $K_p=5$, $K_i=0$, and $K_d=0$	37

5.3	Result of the altitude flight with PID constants (a) $K_p=5$, $K_i=0.01$, $K_d=0$, (b) $K_p=5$, $K_i=1$, $K_d=0$	38
5.4	Result of the altitude flight with PID constants $K_p=10$, $K_i=0$, and $K_d=0$	39
5.5	Result of the altitude flight with PID constants $K_p=20$, $K_i=0$, and $K_d=0$	39
5.6	Result of the altitude flight with PID constants $K_p=20$, $K_i=4$, and $K_d=10$	40
5.7	Result of the altitude flight with PID constants $K_p=15$, $K_i=4$, and $K_d=10$	40
5.8	Result of the yaw controller flight. In the graph, along the red arrow and red lines, the system was disturbed from the equilibrium point by external forces. (a) $K_p=1$, $K_i=0$, $K_d=0$, (b) $K_p=5$, $K_i=0$, $K_d=0$	41
5.9	Result of the GPS controller flight with PID constants $K_p=70$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position . .	43
5.10	Result of the GPS controller flight with PID constants $K_p=35$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position . .	44
5.11	Different reference signals utilized in the position controller loop .	45
5.12	Result of the GPS controller flight with PID constants $K_p=35$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position . .	45
5.13	Result of the GPS controller flight with PID constants $K_p=15$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position . .	46
5.14	Result of the GPS controller flight with PID constants $K_p=5$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position . .	47

5.15	Result of the updated GPS controller flight with PID constants $K_p=30$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position	49
5.16	Result of the updated GPS controller flight with PID constants $K_p=25$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position	50
5.17	Result of the updated GPS controller flight with PID constants $K_p=35$, $K_i=0$, $K_d=0.1$. (a) Error vs. time, (b) x position vs. y position	51
5.18	Result of the updated GPS controller flight with PID constants $K_p=35$, $K_i=0$, $K_d=1$. (a) Error vs. time, (b) x position vs. y position	52
5.19	(a) The results of the experiment using exactly the same controller and filter structure immediately after the indoor flight tests, (b) The results of the outdoor flight after the mentioned adjustments and finetuning of the controller constants were made	53
5.20	(a) The performance of the robot's altitude controller with autonomous take-off and landing, (b) The path following performance of the fully autonomous robot with autonomous take-off and landing	55
5.21	Demonstrates the operation of the flight planner and path generation	56
5.22	Ability of a single robot to follow the created path	58
5.23	Step-by-step demonstration of the obstacle detection system . . .	59
5.24	The results of sensor readings during the test flights with different speeds. Dashed lines represent the thresholds for corresponding sensors. (a) 0.1 m/s, (b) 0.2 m/s, (c) 0.5 m/s, (d) 0.9 m/s	60
5.25	Experimentation environment	62

5.26	Data exchange between subsystems (The direction of the arrows indicates the direction of data transfer. Dotted arrows indicate wireless connectivity.)	64
5.27	(a) Robot's impact moment. As can be seen it does not flex and does not dampen the impact. (b) Robot flips toward the wall and the front directly facing downwards.	67
5.28	The robot with foldable bumper and foldable body crashing. (a) Bumpers flex during the crash, (b) Propellers momentarily stop	68
5.29	The robot with foldable bumper and foldable body crashing. (a) Bumpers flex during the crash, (b) Propellers momentarily stop	69
A.1	Simple PID controller with $K_p=0.001$, $K_i=0$, $K_d=0$	77
A.2	Simple PID controller with $K_p=0.01$, $K_i=0$, $K_d=0$	78
A.3	Simple PID controller with $K_p=0.1$, $K_i=0$, $K_d=0$	78
A.4	Simple PID controller with $K_p=1$, $K_i=0$, $K_d=0.1$	79
A.5	Simple PID controller with $K_p=1$, $K_i=0$, $K_d=0.5$	79
A.6	Simple PID controller with $K_p=1$, $K_i=0$, $K_d=0$	80
A.7	Simple PID controller with $K_p=1$, $K_i=0$, $K_d=1$	80
A.8	Cascaded PID controller with $K_p^0=1$, $K_i^0=20$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)	81
A.9	Cascaded PID controller with $K_p^0=10$, $K_i^0=20$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)	81
A.10	Cascaded PID controller with $K_p^0=15.2$, $K_i^0=10$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)	82

A.11 Cascaded PID controller with $K_p^0=15.2$, $K_i^0=20$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)	82
A.12 Cascaded PID controller with $K_p^0=30$, $K_i^0=0.1$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)	83
A.13 Cascaded PID controller with $K_p^0=40$, $K_i^0=1$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)	83
A.14 Cascaded PID controller with $K_p^0=50$, $K_i^0=1$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)	84
A.15 Cascaded PID controller with $K_p^0=50$, $K_i^0=10$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)	84
A.16 Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.01$, $K_d=0.05$. (Angle is multiplied by 100)	85
A.17 Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.1$, $K_d=0.05$. (Angle is multiplied by 100)	85
A.18 Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.02$, $K_d=0.05$. (Angle is multiplied by 100)	86
A.19 Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.05$, $K_d=0.05$. (Angle is multiplied by 100)	86
A.20 Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.015$, $K_d=0.05$. (Angle is multiplied by 100)	87

List of Tables

5.1	PID constants used in some of the experiments	35
-----	---	----

Chapter 1

Introduction

1.1 Motivation

Automating repetitive tasks has been a research task for a long time to put human power to much more effective use. In this regard, there has been extensive research to automate different tasks, from object manipulation in factory lines to home cleaning tasks via robotic systems. Especially in the last few decades, in addition to the automation of repetitive tasks, robots have been given different abilities to accomplish more delicate tasks, which are dangerous for humans. For instance, stationary robots in factories are able to understand the difference between several objects via different sensors and machine learning algorithms. In contrast, mobile robots can now perform precise localization to move freely and autonomously in the open world (even without GPS). Even though many different wheeled mobile robot designs are very good at what they are planned to do, in such a complicated environment as the world we live in, there is an undeniable need to use legged or flying robots to move on rough terrain or overcome obstacles like stairs. In this sense, using flying robots for inspection of infrastructures such as buildings, bridges, or any hard-to-reach places, as well as cave exploration, is both highly time-efficient and safe for humans because of the agility of flying robots and the need to be in such dangerous places in person is eliminated.

As much as such flying robotic platforms seem to answer all the problems, they are far from perfect, mainly because of their inability to function for a long time and their highly delicate nature. Like most mobile robots, flying robots (i.e., unmanned aerial vehicles) depend solely on a battery on board, which tends to drain extremely quickly compared to terrestrial mobile robots. The delicate nature of unmanned aerial vehicles (UAVs) is also a significant challenge for them to be extensively used. Under optimal circumstances, UAVs are pretty straightforward to fly; however, many different environmental effects may cause a UAV to malfunction and crash. During indoor flights, tight spaces, complex environments crowded with objects, people, and signal interference will likely cause the robot to crash. As for outdoor flights, weather conditions, as well as the complex environment and signal interference, may cause the robot to fail.

Despite their short flight time and easy-to-break nature, unmanned aerial vehicles (UAVs) are beneficial systems as they surely decrease the time required to complete a task and, most importantly, eliminate the dangers to humans. It may be highly challenging to protect the UAV against all of these; creating a UAV platform that is compliant enough to withstand impacts without crashing is a brilliant way to make UAVs much more reliable in complex situations in the real world. Resilience to impacts and crashes is the core purpose of the research on flexible UAVs, as it addresses one of the most significant drawbacks of using a drone.

1.2 Research Objectives

The concept of UAVs, as we know them today, goes back several decades, while many commercial drone companies have formed over the past two decades. The establishment of such companies has relied heavily on the research done prior to the commercialization of drones like quadcopters. Since there has been comprehensive research on drones and quadcopters for many years, the research area is narrow, posing significant challenges to making novel contributions to the existing literature. Nevertheless, this does not mean drones/quadcopters are

in their flawless state, and in fact, they are far from perfect because of their susceptibility to impacts, especially on a smaller scale. The leading cause of the vulnerable nature of drones is their high agility and rigid structure. Because drones can achieve very high speeds compared to any terrestrial mobile robot, their rigidity becomes a critical disadvantage when subjected to impacts. Any collision during the flight will likely result in the complete destruction of the drone body by either falling to the ground or the impact itself. However, the rigid structure has its perks, such as ease of control, a lower computational requirement for prediction algorithms, and keeping body/motor vibrations manageable.

To address this issue, various methods have been proposed throughout the years, including but not limited to using bumpers to protect the propellers, using a cage-like structure around the drone, introducing compliance to the cage, and using propellers with joints to prevent breakage. While all these methods provide protection to drones to some extent, each one has its limitations concerning their application. For instance, cages significantly increase the drone's weight, whereas propeller joints reduce propeller efficiency. Consequently, this work presents a compliant quadcopter body capable of recovering from impacts and crashes.

The main objective of this thesis is to demonstrate the benefits and limitations of introducing compliance to the different parts of the drone body as well as the bumpers. Rigidity and compliance are introduced to the various parts of the drone in different combinations. These combinations include rigid body-rigid bumpers, rigid body-compliant bumpers, compliant body-rigid bumpers, and compliant body-compliant bumpers. The experiments in this study involve impact tests with different speeds for every combination. These impact tests range from 0 to the speed at which the drone crashes or breaks. In this thesis, the body-bumper combination that can sustain flight after the impact with the highest speed is determined. A comprehensive discussion is provided on the collision behavior of every robot with different body bumper combinations.

In addition, with the increasing utilization of Micro Aerial Vehicles (MAVs) in confined and obstacle-rich environments, detecting and controlling the physical interaction between Micro Aerial Vehicles (MAVs) and their surroundings

has become inevitable. Operating in environments with fully/partially unknown maps makes it difficult for collision-avoidance algorithms to avert impacts. Furthermore, these algorithms are computationally expensive and require various sensors, which makes it unfeasible for a very small, fully autonomous standalone MAV to execute them onboard. As a result, various collision-resilient MAVs have been introduced in recent years using advanced materials and fabrication techniques.

1.3 Structure of Thesis

This thesis extensively analyzes and discusses the design, electronics, control, and experiments of a miniature, foldable, collision-resilient quadcopter.

In Chapter 2, the design procedure, from the material selection to the inspiration of the design, is explained. The updates made on the design in relation to the experiment results are also included in this chapter. Together with the body and bumper design of the robot, an impact-detecting soft sensor design is also briefly explained, which is utilized to give the robot the ability to sense its environment.

Chapter 3 of this thesis guides through the step-by-step design process of the most recent PCB used on the robot and the components required to sustain a stable flight. The PCB design is an iterative process for which a few different designs were studied and experimented.

Chapter 4 explains how the control strategy developed during the testing phase. Low-level attitude controllers and high-level controllers such as yaw angle, altitude, and position controllers are explicitly elaborated on in this chapter.

Finally, in Chapter 5, all the results from different designs to new control loops are demonstrated and examined. How controller constants were determined for the attitude controller as well as the experimental results of high-level controllers,

are explained. Results of the soft sensor tests with *Sensorized Collision Resilient aerial Robot (SCoReR)* and impact tests of compliant, rigid, and partially compliant drone bodies are also included.



Chapter 2

Design

2.1 Design of the Robot

As the first step towards making an MAV collision-resilient, it should be ensured that any contact between the drone and its surrounding obstacles will not cause any permanent damage keeping the drone from operating. Not breaking during collisions can be achieved by installing protective structures around the MAV [1, 2]. Utilizing compliance on the protective parts [3, 4] or the whole drone frame [5] will increase its impact tolerance by dampening the impact forces. The MAV frame can also be designed to demonstrate variable stiffness in different situations, being rigid in the absence of collisions and compliant in their presence [6]. Origami-inspired structures seem to be reasonable choices for increasing the impact-resilience properties of MAVs because of being lightweight and dissipating high energy during impacts [1, 3, 4, 5, 7].

The proposed flying robot body design will be manufactured using the foldable robotics technique. The basis of the technique is the production of three-dimensional structures from a single flexible sheet in a two-dimensional manner with the help of a laser cutter, and then folding this flexible sheet to obtain three-dimensional structures. In addition, the production technique used in this

project has emerged with the addition of origami-inspired parts in the design phase. The amount of rigidity of the places on the body where flexibility is desired, is controlled by geometric parameters.

The films such as cellulose acetate or PET sheets are used in production. The body is symmetrically designed to distribute weight evenly. A large number of body designs are planned, with varying flexibility and dimensions during the manufacturing phase; the foldable robotics technique is well suited for this project, as the structures can be produced, and the designs can be updated quickly.

For the final design of the collision-resilient quadcopter body, the absolute maximum dimensions are determined to be smaller than 300 mm in every dimension while keeping the body below 300 grams. However, it is aimed to keep the robot as small as possible with a payload capacity of at least 250 grams as the robots needs to carry around 183.5 grams of payload as the electronics, battery and motors.

Keeping the compliant robot’s manufacturing costs low by using easily accessible materials on the market and traditional manufacturing techniques is one aim of this research. PET (Polyethylene Terephthalate) sheets with a thickness of 0.25 mm are used to create the foldable components. Office stores and print industries often stock these sheets in various colors and thicknesses. The material’s mechanical properties also make it an excellent option for creating origami structures. The folding lines and origami patterns are engraved on the PET sheets using a desktop laser cutter. A detailed explanation of the design and manufacturing of the foldable arms, central housing, protective bumpers, landing gear, battery housing, and camera casing is given in [8].

In addition, a *Sensorized Collision Resilient aerial Robot (SCoReR)* is designed to detect collisions via a soft spring-shaped sensor. The collision-resilient frame of SCoReR is developed based on our previous design (MiniCoRe [5]) using origami-inspired folding techniques. Different parts are manufactured by folding a 2D pattern (cut from a single A4 size PET sheet) into a 3D structure. A commercial laser engraver (Universal Laser Systems VLS 3.50) is used to cut

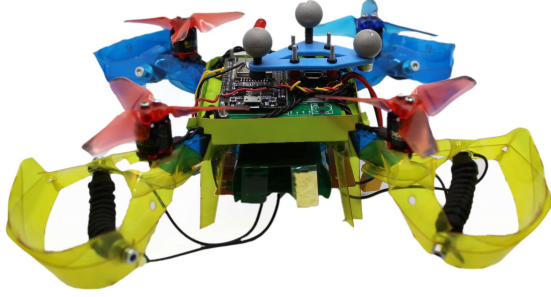


Figure 2.1: *Sensorized Collision Resilient aerial Robot (SCoReR)*

the patterns from the sheets. Overall, there are twelve laser-cut folded parts in SCoReR:

- Two T-shaped arms, folded from two symmetric triangular structures attached in the middle by a groove-and-tongue joint, form an X-shaped quadcopter frame. There are five pairs of extension locks on each side of these four arms (twenty extension locks in total). Other folded components are mounted on the frame using these locks. Motor mounts are designed as extensions at the arm tips (Figure 2.2).
- A central housing is mounted on the arms using sixteen extension lock pairs in total (four locks on each arm).
- Four landing gears are installed on a single lock on each arm, keeping the arms in tension and preventing them from excessive vibrations and twisting during the flight (Figure 2.3(a)).
- A battery housing is mounted underneath the arms by four pairs of extension locks.
- Four protective bumpers are screwed between the folded motor mounts and the motors. The contact face of the bumpers is designed to be inclined (angle γ illustrated in Figure 2.3(b)) to passively compensate for the rolling/pitching moment during the collision with a flat vertical surface. The contact-detecting soft sensors are also mounted on the slots embedded in the perimeter of the protective bumpers.

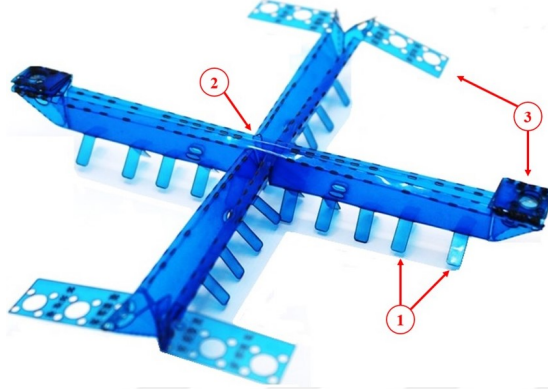
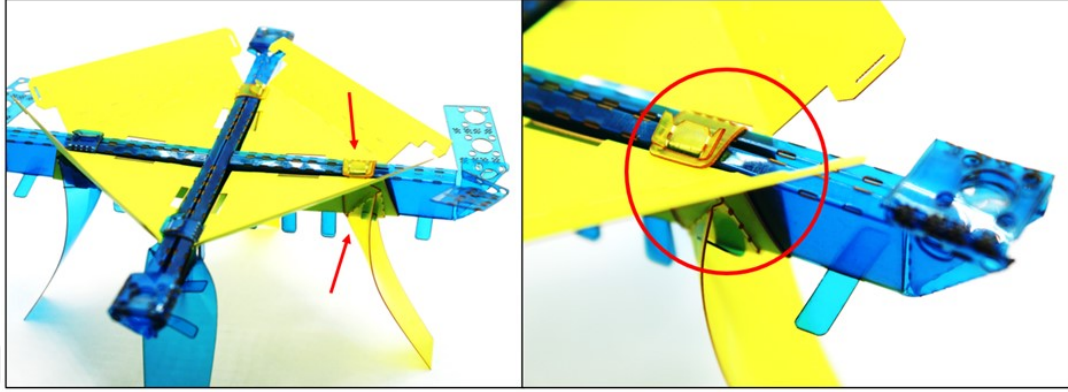


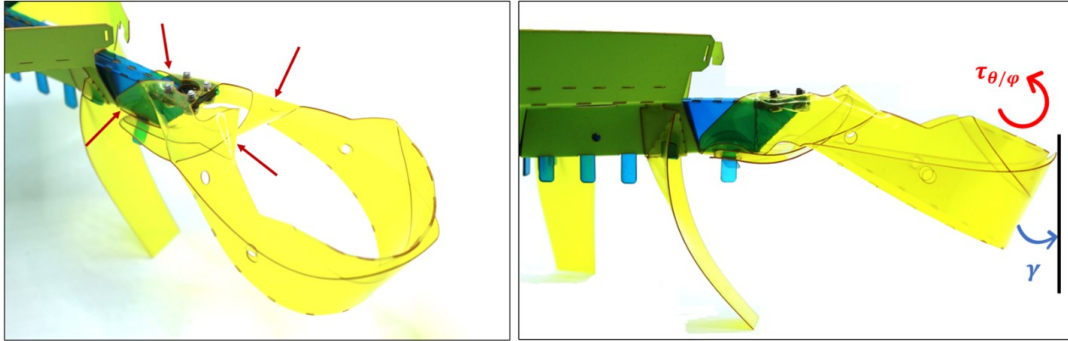
Figure 2.2: Foldable arm design; the extension locks (1), the groove-and-tongue joint (2), and the folded and unfolded motor mounts (3)

To attach the quadcopter’s electronics to the foldable frame, we use an intermediate part fabricated from TPU (Thermoplastic polyurethane) filaments (soft material) using an FDM 3D printer. This part holds the main PCB (Printed Circuit Board) through four screws on the top and is bolted on the arms through four fork-like extensions at the bottom. The final quadcopter body has maximum dimensions of $19\text{cm} \times 19\text{cm} \times 9\text{cm}$, weighing approximately 220g, whereas the foldable body constitutes 36.5g of total weight. More details about the foldable body design for SCoReR and the component selection can be found in [8].

Besides the SCoReR design, to increase the maximum possible impact speed that the robot can endure without crashing on the ground, the design of the foldable bumpers has been updated. The updated design is shown in the Figure 2.4. With the previous design of the bumpers, the robot almost always failed because the bumpers kept squishing during impact, and the propellers hit the obstacle. The core purpose of the design change is to increase the resistance of the bumpers during a collision so that propellers would hit the obstacle at a higher speed. How the new design affected the impact characteristic of the drone and the impact test results are explained in Chapter 5 of this thesis.



(a)



(b)

Figure 2.3: (a) Landing gear installation on the central housing and arms (b) Illustration of the foldable bumper installation on the arms. Four extensions are fastened on the last lock on the arm (left), the inclination of the bumpers help the drone recover about the pitching/rolling axis in collisions (right)

2.2 Sensor Design

The second step of collision resilience is to detect the impacts during the flight and prevent the impacts from reoccurring. Although some researchers introduced, impact resilience methods using passive mechanisms [9], others tried to detect physical contact using various methods. External odometry (such as motion tracking systems) can be used to detect a possible collision [10]. Onboard sensors can also be used to detect the contacts/collisions on a MAV, independent of any calculations run on an external processor. Authors in [11] introduce a

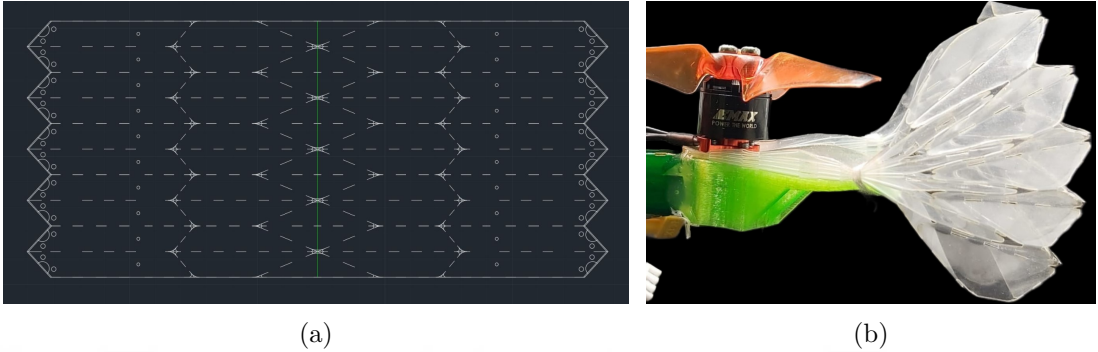


Figure 2.4: (a) Unfolded design of the new bumper design, (b) Folded and mounted version of the new bumper design

small force sensor (based on Hall sensors) placed on the pivot joint of a cage-like structure around the drone. Using eight of these sensors around the drone, they detect the impacts. The exact mechanism (Hall sensors) is used in [12] in larger dimensions, and more compliance is introduced around the body for collision resilience. However, this solution does not seem to be easily scalable for miniature-size MAVs. Using only Inertial Measurement Units (IMUs) and cameras, the authors in [13] have done a fantastic job of detecting possible contacts of an MAV without physically sensing it. In this study, we introduce an alternative method to the latter, where soft tactile sensors are used to detect contacts/impacts to an obstacle.

The soft sensors embedded in SCoReR bumpers are similar to those utilized in [14]. The soft sensors are placed on the foldable bumpers on SCoReR to detect the possible obstacles the drone may encounter. We expected to capture the impact of the collision with the soft sensors.

When the soft coil-spring sensor coils collapse, we observe a significant drop in sensor resistance because adjacent coils of the sensor contact each other [14]. As a result, keeping the distance between the coils of the coil-spring-shaped soft sensors as low as possible makes the sensor more sensitive to impact forces.

A screw-type electrical connection is designed for the soft sensors to prevent noise caused by the vibration during the flight and the propellers' airflow.



Figure 2.5: (a) CAD model of the soft sensor, (b) Picture of the soft sensor

Threaded inserts are placed on both ends of the soft sensor to fabricate the screw-type electrical connectors, as shown in Figure 2.5(b). The connection cables for the sensors are soldered to these inserts. Soft sensors can be attached to the foldable bumpers using screws using the holes created on the bumpers, as shown in Figure 2.1. The nominal resistance of the soft sensor (in the non-compressed form) is in the range of 130-150k Ω . The sensors buckle and compress when they deform. As a result, the resistance of the sensors decreases by 75%. 3D printing settings of the soft sensors for SCoReR are the same as the ones presented in [14].

Chapter 3

Electronics

For a drone to sustain stable flight, there are several electronic components and hardware required to be used, including but not limited to motors, electronic speed controllers (ESCs), a microcontroller unit (MCU), and a flight controller (FC). This chapter describes all the components, their corresponding selection processes, and the design process of the printed circuit board (PCB), which connects every element of the quadcopter.

3.1 Component Selection

Electronic equipment on a drone may vary, considering how the drone will be utilized and the needs of the duties that it will be carried out. Onboard equipment may include an RGB-Depth camera [15], optical flow sensors [16], lidar [17], barometer and even soft sensors [18]. However, every drone with a multirotor structure must have four main components onboard: motors, ESCs, MCU, and a flight controller.

The component selection process started with approximating the required lift per motor to choose an appropriate motor and propeller combination. We chose a combination of a motor and a propeller as, in general, most modern brushless

DC motors are explicitly designed to work more efficiently with a specific type of propeller. Emax RS1106II BLDC with EMAX AVAN MINI 3-inch propellers are selected because the motors have better internal cooling to prevent burning. Also, Obtaining them was very convenient as they could be found within the country. A suitable ESC must be added to the system for the motor and propeller combination to function correctly. Instead of getting the ESC recommended for the motors, we decided to use a flight controller, which includes a four-in-one ESC on board, fitting neater on our design and occupying much less space. Thus, GepRC Stable F411 ESC and FC Stack are used on the robot to control the drone's attitude. Finally, ESP32 is the microcontroller unit for altitude and position controllers and communication. From the variety of different microcontrollers, we decide upon ESP32 as it has an onboard wireless communication module supporting ESP-NOW protocol which we use to communicate with the motion capture system. It was also helpful that ESP32 has a relatively high clock speed and two physical cores.

3.2 PCB Design

All the abovementioned electronic components require a stable connection with each other to function correctly. Hence, a PCB which connects every element is a necessity. Throughout this project, eight different designs have been prepared, and the key designs and their differences will be explained in this section.

The first PCB we designed and used was to conduct experiments and test the attitude controller we designed and programmed. As the attitude controller structure is explained in detail in Chapter 4 of this thesis, the robot requires to receive data about its current state from IMU, which is located on the PCB. Furthermore, the barometer to measure the altitude of the robot is also included, just like the IMU, so that it could easily be utilized in the control loops in the future. In addition to the sensors, the main controller unit, ESP32, is located in the middle as well as the ESC connections on the corners of the PCB. Every electronic component works with 5V, which the ESC supplies with its internal

regulator. However, in case it is needed to add an extra regulator or any device requiring 5V, there are extra 5V and ground connection pins on the PCB. As seen in Figure 3.1, the PCB is relatively large, with much empty space. The open area on the PCB makes it possible to place various sensors without changing the PCB design, as we were testing different sensors at the time. So, in case of sensor change, even the larger sensors can be tested without any change in the PCB design, thanks to the empty space.

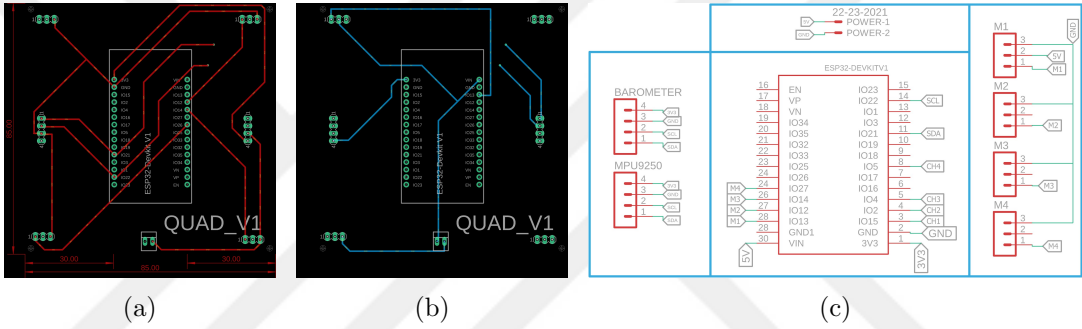


Figure 3.1: First version of the PCB used on the robot. (a) Top view, (b) Bottom view, (c) Schematics of the PCB

In addition to this design, after completing several attitude controller experiments detailed in Chapter 5, we switched the robot from employing the attitude controller we programmed to a much more stable ready-to-fly flight controller. The brand of the flight controller is GepRC Stable F411, and we send PPM signals to control the drone. For this purpose, the GepRC flight controller needed to be added to the PCB, so a new PCB was required. In addition to ESP32, IMU, and barometer, a UART port is added to the newer PCB for the GPS, which is used during outdoor flight tests, as explained in [8]. The ESC connections are removed since GepRC has a four-in-one ESC on board. Finally, a voltage regulator and two compasses are added to the newer design. Besides the added components to the system, the secondary objective of designing such a PCB is to make it much more compact, so two different designs are prepared. One of the designs has every sensor as a module having its own PCB. On the other hand, the secondary one, which uses solely the sensor chips, is designed and produced to form the smallest PCB possible. Both designs can be seen in Figure 3.2 and the marked components onboard.

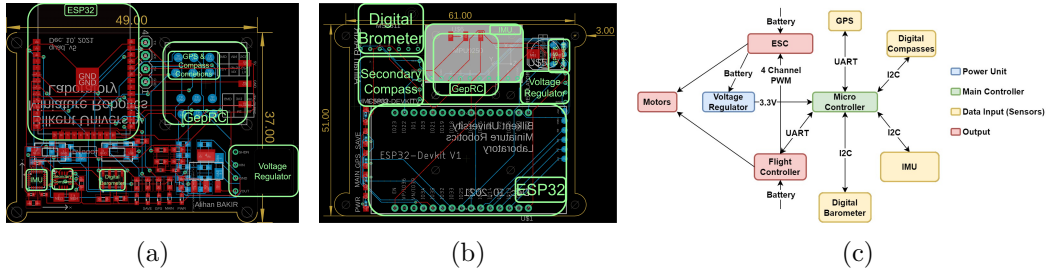


Figure 3.2: Two separate PCB designs with same connection diagram. (a) The one with sensor chips, (b) The one with sensors as modules, (c) Connection diagram of both PCBs

The final design of the PCB on the robot is only required because of the need for extra components for the experiments presented in [18]. Four Wheatstone bridges were necessarily put on the PCB because of the addition of soft spring-shaped sensors to the bumpers in [18] to detect collisions. With the Wheatstone bridges, additional connectivity exists for extra sensors and debugging purposes. Additionally, all the power lines were made thicker in the newer design because of the high power consumption of the Wi-Fi and Bluetooth module of the ESP32. As a result, the drone now has an all-in-one PCB encompassing every necessary component for further development and studies. The final version of the PCB can be seen in Figure 3.3.

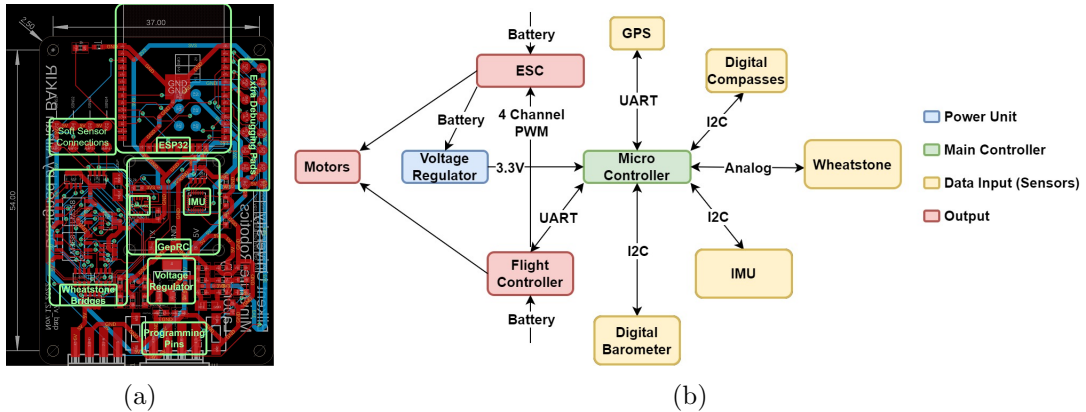


Figure 3.3: Final version of the PCB used on the robot. (a) Board view, (b) Connection diagram of the PCB

Chapter 4

Control and Operation

The project and related work explained in this thesis may be divided into two main parts according to the stage of the whole work. During the first stage, a couple of different controller designs were tested. A simple PID loop and a cascaded PID loop structure with several different filters and estimators have been experimented with, which are explained in detail in the Attitude Controller section of this chapter and the Experiments and Results chapter. At this stage of the project, studies are carried out to ensure that the designed UAV has a stable flight. In this project, since the tasks defined for the UAV are visual inspection and swarm movement, they must be controlled in a stabilized manner (not in race (or acro) mode). The main difference between the two is that in stabilized motion, the controller allows the UAV to follow user-defined angles (Euler angles), but in acro motion, the controller allows the UAV to follow user-defined angular velocities. Therefore, the attitude controller is designed and programmed for further testing, which adopts a cascaded PID loop structure as in [1] and [2]. For the next stage of the project, the attitude controller is switched from the one designed in the first stage to a ready-to-fly flight controller, GepRC, for better stability and control. Even though both have the same cascaded PID loop structure, GepRC engenders a much more stable flight thanks to its higher loop and data acquisition speed combined with better filters and estimators, which are not disclosed. Moreover, a position controller, a yaw angle controller, and

an altitude controller are integrated with GepRC, which are executed in the ESP32 microcontroller. These controllers are analyzed in detail in the “outer-loop controllers” section of this chapter.

4.1 Attitude Controller

The attitude controller of a drone is a low-level controller, including angular position and velocity controllers for roll and pitch angles as well as the angular velocity controller for the yaw angle. There are various controller designs, from linear PID loops for every DoF of the system to state space controllers like in [19]. Even though state space controllers require higher computational power onboard, they generally offer a much smoother flight. On the other hand, a PID controller or cascaded PID controller provides a stable flight with much lower requirements.

Considering the power requirements and ease of implementation, we started experimenting with a simple PID controller, which can be seen in Figure 4.1. A similar approach has been considered previously in the literature, mainly because of its simplicity [20, 21]. The PID loop receives data from the IMU onboard, which includes 6 DoF gyro and accelerometer measurements. The gyro measures angular speeds around the x, y, and z axes, while the accelerometer measures the linear acceleration in the x, y, and z axes totaling 6 DoF. The measurements for the gyro and accelerometer are received by the main controller on board, ESP32, via an I2C connection at 50 Hz. The integral of the gyro data is combined with the angle of the acceleration vector via a complementary filter to calculate the angular position of the robot explicitly for pitch and roll angles. The complementary filter, whose equation is given in equation 4.1, is purposefully utilized in this PID loop primarily because of its low computational demand, comprising simple multiplication and addition. The calculated angular positions are then used to find the error. The PID controller’s proportional, integral, and derivative terms are computed with the error and fed into the ESCs as PWM signals which directly control the motors. The same loop structure is used for the yaw controller with only one difference; instead of angular position as feedback and references,

angular speed is used. The robot could not sustain a stable flight using the linear PID loop, as discussed in Chapter 5, so we were compelled to change the whole controller structure.

$$\phi = \omega_\alpha * \phi_{accel} + \omega_g * \phi_{gyro} \quad (4.1a)$$

$$\theta = \omega_\alpha * \theta_{accel} + \omega_g * \theta_{gyro} \quad (4.1b)$$

In this equation set, ϕ is the roll angle, and θ is the pitch angle of the robot. ω_α determines the weight of the calculated angle from acceleration measurements, and ω_g determines the weight of the angle calculation from gyro data. Adding ω_α and ω_g must add up to one for the filter to work. ϕ_{accel} and ϕ_{gyro} represent the calculated roll angle from the accelerometer and gyro data, respectively. Similarly, θ_{accel} is the pitch angle from acceleration data, and θ_{gyro} is the pitch angle from gyro data.

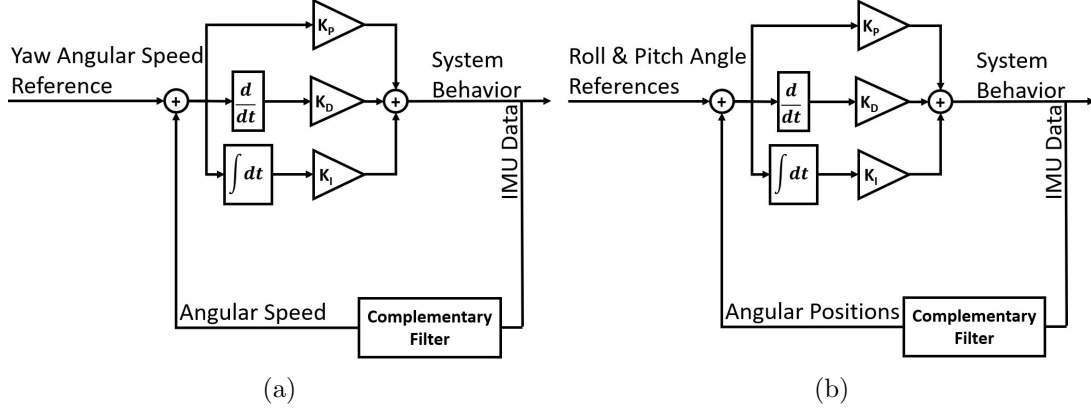


Figure 4.1: Simple PID controller used in the initial flight tests. (a) For roll and pitch angles, (b) For yaw angle

Furthermore, the robot's attitude controller is switched from a simple PID controller to a cascaded PID controller to sustain a stable flight, which is much more reliable for drones in general. There are invaluable collections of research that are already done in a very comprehensive way, and a substantial amount of them acknowledge that the cascaded PID loop is a reasonably stable controller for a drone [22, 23, 24]. Because of the instability issues with the simple PID controller, a cascaded PID loop is employed in the robot for further testing. The

cascaded PID controller comprises two simple PID loops where the output of one is the reference input of the next. Just like the abovementioned simple PID loop, the feedback signal is calculated from 6 DoF IMU data via two separate complementary filters; one for angular rates and the other for angular positions. The former is sent to the inner loop as a feedback signal, while the latter is the feedback signal of the outer loop. For roll and pitch angles, the reference is generated by the user at this stage of the project (during autonomous flight, they come from the position controller), and for the yaw controller, angular rate references are generated by the user. Roll and pitch error is fed into the PID controller, which is calculated with the references and the feedback. The output of the first PID is sent to the inner PID as its angular rate references. The rate error is sent to the inner PID after being calculated via the rate feedback and the references. The output of the cascaded PID loop is directly sent to the ESC as a PWM signal for every motor.

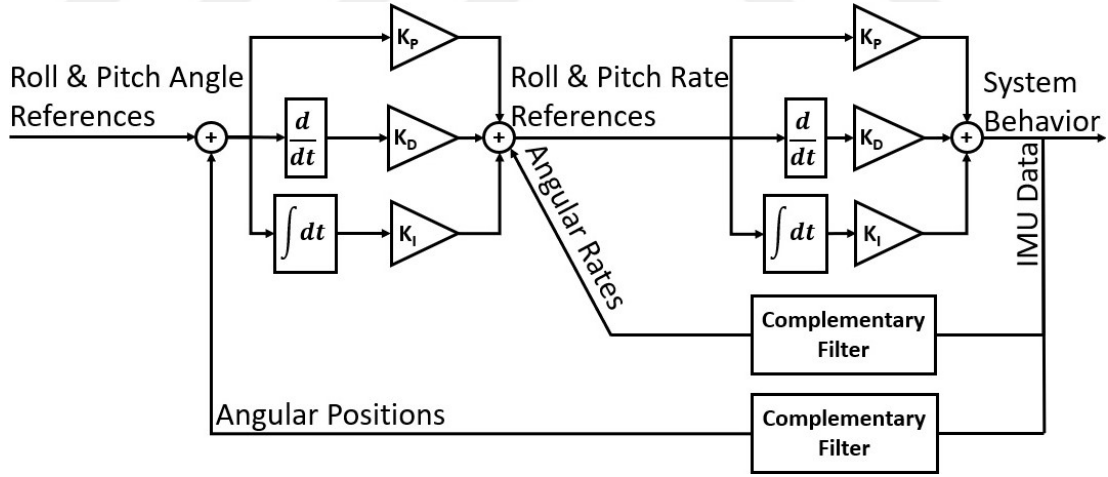


Figure 4.2: Cascaded PID loop schematics for roll and pitch angle control

After testing several control strategies for attitude control of the drone, the cascaded PID loop for roll and pitch angle control and yaw rate control presents better stability, for which detailed data are given in Appendix A as well as a profound analysis in Chapter 5. During the final test flights, the robot is able to lift off and fly following the user's commands. However, it would have been particularly challenging, and it is not straightforward to control even for people

because the system still has vibrations. Because of this, GepRC, a ready-to-fly flight controller, is used as a low-level attitude controller. Due to a large number of commercially available ready-to-fly flight controllers employing the same controller structure as the one in our robot, data acquisition frequency and data filtering are in question. The GepRC flight controller runs the control loop at 8 kHz in default settings and can go up to 32 kHz [25]. On the other hand, our control loop is only run at 50 Hz. The data acquisition frequency is also 50 Hz in our program in spite of the GepRC receiving data from the accelerometer at 8kHz and the gyro at 2 kHz [25]. When the frequency discrepancy in the control loop and data acquisition is combined with more sophisticated filtering algorithms, which are not disclosed, GepRC offers a much more stable flight.

4.2 High-Level Controllers

In addition to the GepRC flight controller, which uses Betaflight software, high-level controllers are included in the system for fully autonomous flight. Together with the higher execution frequency, GepRC offers a truly stable flight. While GepRC executes the attitude controller of the robot, the ESP32 that we programmed runs the high-level controllers such as the yaw angle controller, altitude controller, and position controller. The high-level control loop we designed sends control references to the GEPRC in line with the data from the outputs of the altitude control loop, yaw angle control loop, and position control loop. In light of the incoming commands, the GEPRC flies the UAV stably on the route specified by the user. The high-level controllers operate at 200 Hz and are analyzed in detail in the following sections. Altogether, these controllers form a drone that is capable of fully autonomous flight. During the design and implementation of the high-level controllers, all of them are designed and added to the system individually as outer loop controllers.

4.2.1 Yaw Angle Controller

IST8310 coded digital compass was used to control the yaw angle. The reasons for choosing the IST8310 digital compass as the heading sensor are explained in detail in [8]. The IST8310 digital compass works with the I2C protocol and is made ready for initial use using I2C commands as specified in the datasheet [26]. All commands that can be sent via the I2C protocol are included in the datasheet [26]. Similarly, data is read from the sensor separately for the x, y, and z axes via the I2C protocol. The read data is converted into Gaussian units so that it can be included in mathematical operations later. The Gaussian data is calibrated by using the function given below with the calibration constants found with the help of Matlab. The angle of the UAV around the z-axis is found by taking the inverse tangent of the calibrated data. After the angle found is also offset so that 0° corresponds to the north direction, the data between 0° and 360° are unwrapped so that they are between negative infinity and positive infinity. The angular data, which is now unwrapped, is fed into the one-dimensional Kalman filter. The resultant data is ready to be used in the control loop as the feedback signal.

$$\begin{bmatrix} x_{calibrated} \\ y_{calibrated} \\ z_{calibrated} \end{bmatrix} = A * \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix} - b \right) \quad (4.2)$$

In this equation, matrix A and vector b were found with the magcal function embedded in Matlab software. x , y , and z are the raw data read from the sensor, and $x_{calibrated}$, $y_{calibrated}$, and $z_{calibrated}$ values are the calibrated sensor data.

One fundamental difference when moving the experiments inside is that the angle controller's outer loop feedback is gathered and calculated from the motion capture instead of the digital compass. From the raw motion capture data, the heading of the quadcopter is calculated as it needs to be corrected and then fed into the outer loop directly. The loop receives yaw angle references during the flight and outputs yaw rate references which are then fed into the GepRC, and the rest of the loop is run as a part of the attitude controller.

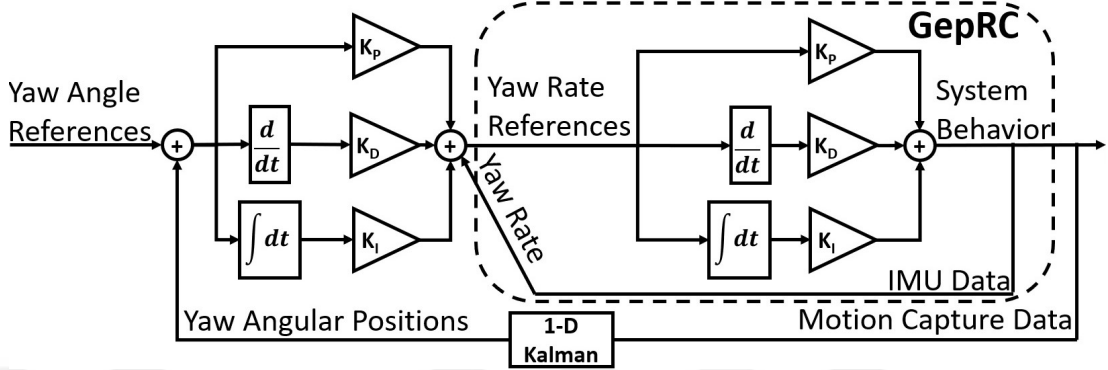


Figure 4.3: Cascaded PID loop schematics for yaw angle control

4.2.2 Altitude Controller

The first step in working with the altitude control loop was to investigate the types of sensors that can be used to measure altitude and select the appropriate sensor type. In line with the examinations and research, it was decided that the digital barometer MS5611 of TE Connectivity brand could be used to calculate the altitude of the UAV above sea level. The first step for altitude control is to connect this sensor to our ESP32 controller, enabling data flow, and detailed information on this subject is given in the Chapter 3 of this thesis. MS5611 digital pressure sensor communicates using I2C protocol. As for many sensors, Measurement frequency and accuracy are inversely proportional for the MS5611 digital barometer. In order to measure the altitude of the UAV as accurately as possible, the measurement frequency and measurement accuracy were determined to maximize the accuracy so that the measurement frequency was set to 100 Hz as described in the datasheet of the sensor. The 16-bit digital data coming in at a constant 100 Hz was converted into millibar (*mBar*) for mathematical calculations. The relative altitude of the UAV with respect to the initial point was calculated with the following formula, using the ground-level air pressure value calculated and recorded just before the start of the flight as a reference.

$$h = 44330 * (1 - (\frac{P_{instant}}{P_{initial}})^{0.1903})[27] \quad (4.3)$$

In this formula, h is the relative altitude of the UAV with respect to the ground

measurements at the initial point, $P_{instantaneous}$ is the instantaneous measurement from the MS5611 digital barometer, and $P_{initial}$ is the reference pressure measurement at the ground level just before the flight. The altitude data obtained from this calculation has an accuracy of $\pm 10\text{cm}$ [27].

Since it is impossible to use the raw relative altitude data in the control loop, the altitude data from the sensor is filtered before entering the altitude control loop. Initially, a weighted moving average was used for filtering, but due to the high noise in the incoming data, a one-dimensional Kalman filter was used. The altitude value obtained from the weighted moving average is calculated using the formula below.

$$h_{filtered} = \alpha * h_{filtered} + (1 - \alpha) * h_{new} \quad (4.4)$$

In this equation, α is the weight of new incoming faulty data. As α increases, the weight of new data decreases, and the operating speed of the filter, i.e., its sensitivity, decreases.

The basic principle of the one-dimensional Kalman filter is the prediction of future data and the continuous updating of this prediction. Three main steps are followed when calculating the estimated altitude using the one-dimensional Kalman filter. These steps start by defining the necessary variables before initializing the sensor reading cycle. The measurement error, prediction error, and process error are the variables used in the Kalman filter. The prediction error is updated in each cycle, so how the initial definition is made is not essential. The measurement error variable is the average error between the actual altitude value and the measurement, and the process error variable determines how much weight to give to the current and erroneous measurement. In other words, the process error variable determines how sensitive the filter is. The more sensitive the filter is and the faster it runs, the higher the noise. The process error variable is generally taken as the standard deviation of the data collected from the system. Once these variables are determined, the operations performed in each measurement cycle are expressed in the following formulas.

$$gain_{Kalman} = \frac{\epsilon_{prediction}}{\epsilon_{prediction} + \epsilon_{measurement}} \quad (4.5a)$$

$$prediction_{current} = prediction_{previous} + gain_{Kalman} * (measurement - prediction_{previous}) \quad (4.5b)$$

$$\epsilon_{prediction} = (1 - gain_{Kalman}) * \epsilon_{prediction} + |prediction_{previous} - prediction_{current}| + q \quad (4.5c)$$

$$prediction_{previous} = prediction_{current} \quad (4.5d)$$

The equation set 4.5 expresses the filtered altitude data using the one-dimensional Kalman filter with the variable $prediction_{current}$, which is repeated and updated at each measurement cycle. The variable $gain_{Kalman}$ is calculated from the error at each measurement cycle. $\epsilon_{prediction}$ is the error in the computed estimate. q is the process error variable.

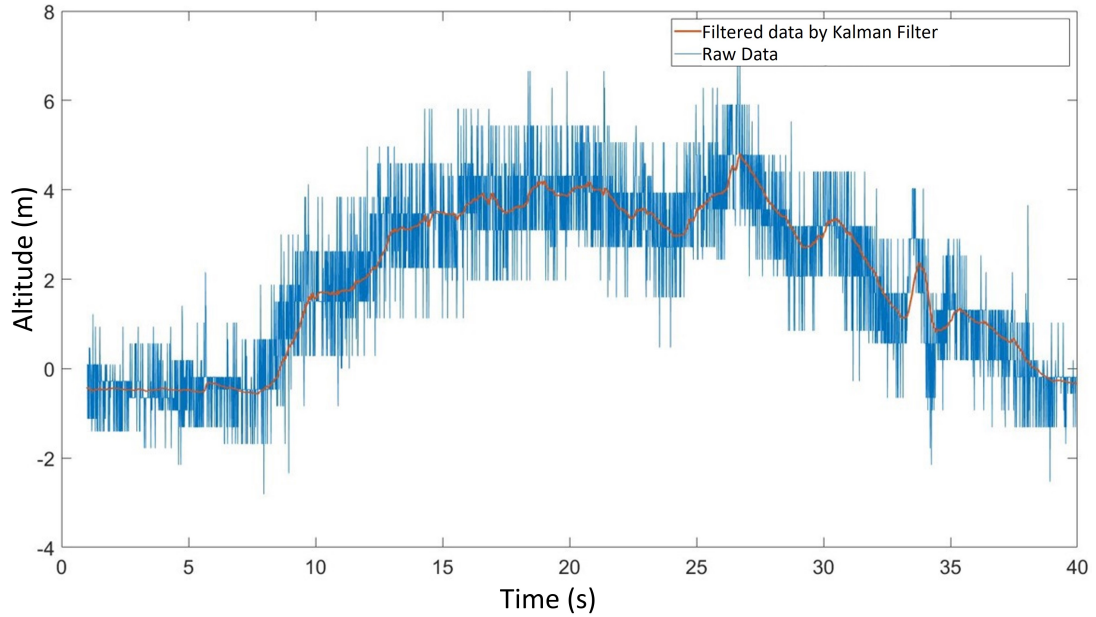


Figure 4.4: 1-D Kalman estimator for digital barometer data

As shown in Figure 4.4, the Kalman Filter works quite stably, which removes noise from the incoming raw altitude data. At the same time, it should be noted that the difference between the altitude measurement at the start of the flight and

the altitude measurement at the end of the flight is almost negligible, meaning that the low-frequency drift is negligible.

After this sequence of mathematical operations is executed, the filtered altitude data is ready to be fed into the control loop. As can be seen from the altitude controller schematic shown below, the altitude reference data, which the user predefines, is sent to the Proportional (P), Integral (I), and Differential (D) controller. The output of this controller is sent directly to the flight controller in the system as the throttle value and the system's response, as the data from the sensor, passes through the Kalman filter again and returns to the altitude controller to be used in the next cycle. This entire process forms the altitude controller. This controller, also called a PID controller, has three separate control parameters: K_p , K_i , and K_d . Changing these parameters results in different behavior of the entire controller and, therefore, the entire system, i.e. the UAV. The parameters K_p , K_i , and K_d are found by trial and error, and the process of finding these parameters is analyzed in detail in the rest of this report.

Although the altitude controller is relatively simple compared to other controllers utilized on the drone, autonomous takeoff and landing also require the use of an altitude controller, which is a more challenging task. Depending on the robot's battery level, the controller output varies to keep the UAV at a certain altitude, and the altitude controller must work around this value to maintain altitude. Since this value changes for each flight and UAV, the UAV needs to calculate it itself during or prior to the flight. In addition, many different methods have been tried to make autonomous takeoff and landing possible. First, the integral term of the altitude controller is initialized with a value greater than zero, and the autonomous landing and takeoff are assigned to the integral term of the altitude controller. The rate of increase of the integral term of the altitude controller is directly proportional to the integral constant (K_i) of the controller. When K_i is low, the controller's integral term increases very slowly, causing the UAV to be unable to take off, and when K_i is high, the UAV becomes unstable after taking off. These problems can be seen in the data in Figure 4.6.

In addition to the controller described above, a nominal throttle value is added

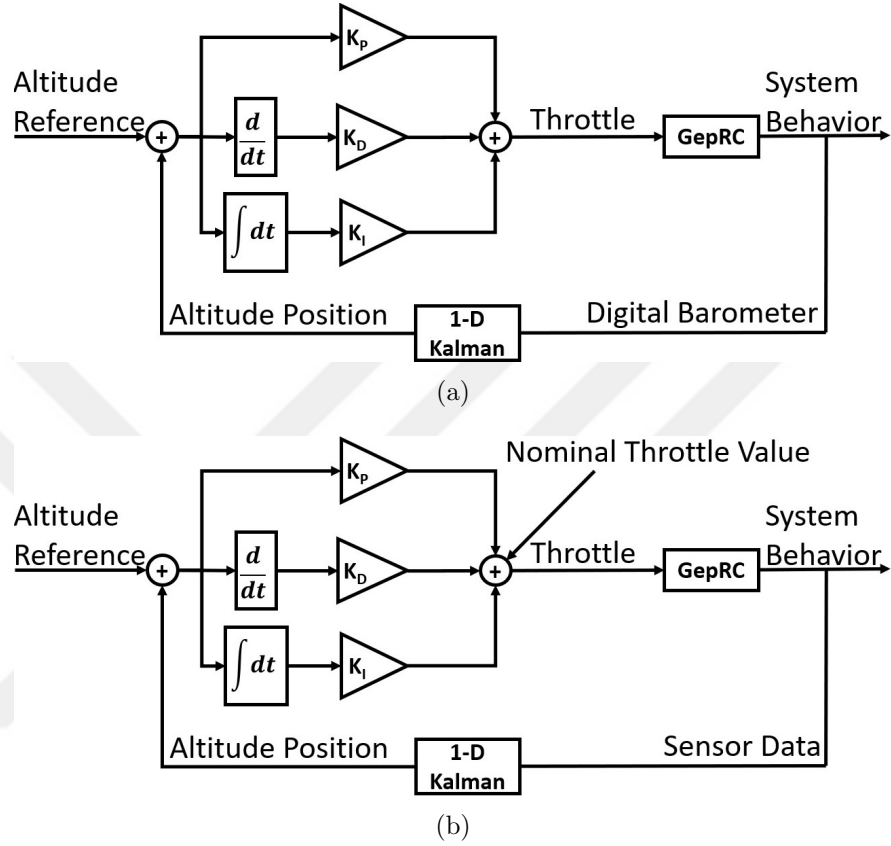
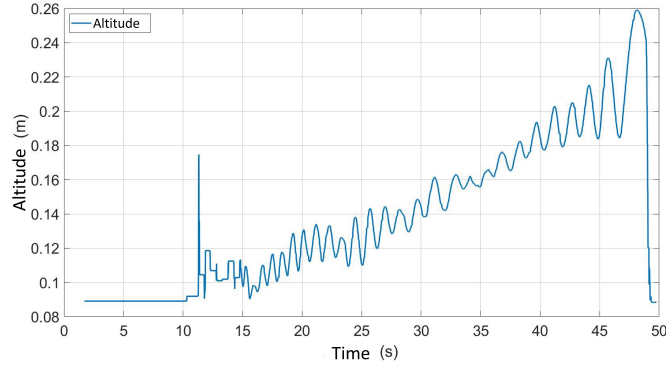


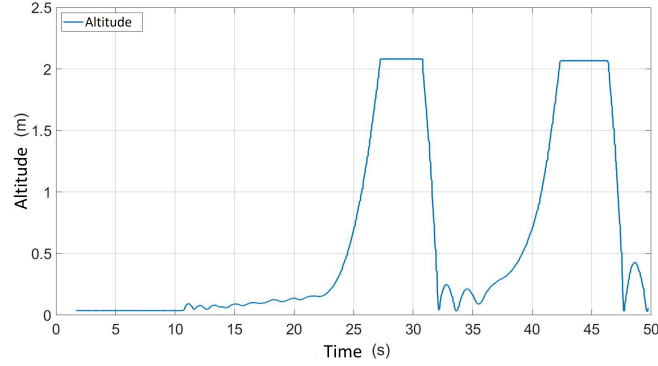
Figure 4.5: (a) First altitude controller with no nominal throttle value, (b) The final version with nominal throttle value. Sensor data is from the digital barometer for outdoor flights and motion capture data for indoor flights

because the control loop is designed to hold the robot around a zero point. However, the throttle value needs to be held around a positive value, so an offset is added, which is the nominal throttle value. The exact number for the nominal throttle is calculated before every flight during takeoff by increasing the throttle value linearly until the drone reaches around 50 cm, and that throttle value is saved as the nominal throttle value. Adding the value directly to the output of the altitude controller results in a controller with a positive offset to control the drone around it. Then the result of the addition is directly fed into the attitude controller, GepRC as the throttle value. The control loop with the nominal throttle value can be seen in Figure 4.5.

The feedback is changed during the experiments for transitioning from outdoor



(a)



(b)

Figure 4.6: (a) When the integral term of the altitude controller is increased from 180 to 230, the UAV cannot take off. (b) When the K_i constant of the altitude controller is increased, the UAV can take off but cannot reach a stable point.

to indoor tests. For indoor testing, the sensor feedback is received from the motion capture system as it is more accurate, especially indoors. Changing the feedback sensor from a digital barometer to a motion capture system requires updating K_p , K_i , and K_d constants.

4.2.3 Position Controller

The position controller is the last controller to be added to enable the fully autonomous flight of the UAV. The sensor for the feedback signal of the position control loop is GPS. Due to the limited space on the UAV and the payload

capacity of the UAV, there are very few GPS sensor options that can be used, for which the selection process is described in detail in [8]. The GPS sensor currently used on the UAV is the M8N model from Ublox. Unlike other sensors, the M8N GPS sensor communicates with the UART protocol rather than the I2C protocol. The M8N sensor can send data at up to 10 Hz, but to make this possible, non-essential data flow is blocked, and the satellites to be connected are limited, as described in the datasheet of the M8N GPS sensor. The data from the M8N sensor to the main controller is in NMEA standards and needs to be decoded to access the actual position data. The sensor data was decoded as directed in the datasheet provided by the sensor manufacturer [28].

Unfortunately, the sensor data cannot be sent directly to the position controller as the feedback since the sensor data are in spherical coordinates in the north and east directions. The reference value of the position controller should be the coordinates of the UAV on the x and y axes relative to a specific point rather than the spherical coordinates. The takeoff point of the UAV is always taken as the origin of these x and y axes so that the controller is not affected by the spherical shape of the Earth, the PID controller can be used, and the linear approximation can be applied. There are several different ways to write the spherical coordinates in the form of x and y coordinates, but since the takeoff point is chosen as the center, the way that is followed is to convert the spherical coordinates to the Earth-centered, Earth-fixed coordinate system (ecef) and from the Earth-centered, Earth-fixed coordinate system to the local tangent plane coordinate system (North, East, Down - ned). The method followed separately for the three axes for the transition from the spherical coordinate system to the geocentric geostationary coordinate system is given in the equation set 4.6.

$$x_{ecef} = \frac{6378137}{\sqrt{1 - \frac{\sin^2 lat}{150}}} * \cos lat * \cos lon [29, 30] \quad (4.6a)$$

$$y_{ecef} = \frac{6378137}{\sqrt{1 - \frac{\sin^2 lat}{150}}} * \cos lat * \sin lon [29, 30] \quad (4.6b)$$

$$z_{ecef} = (1 - 0.00669438) * \frac{6378137}{\sqrt{1 - \frac{\sin^2 lat}{150}}} * \sin lat [29, 30] \quad (4.6c)$$

In the set of equations above, x_{ecef} , y_{ecef} , and z_{ecef} represent the x, y, and z components of the UAV's position, respectively, relative to the earth-centered Earth-fixed coordinate system, while lat and lon represent latitude and longitude.

The transition from the Earth-centered Earth-fixed coordinate system to the local tangent plane Coordinate System (North, East, Down - NED) is completed with a slightly more straightforward set of mathematical equations since there are no complex effects such as the shape of the Earth. Unlike the previous transformation, the transformation from the ECEF frame to the NED frame, for which the mathematical equations are given below, requires reference data in addition to the current data. The reference data is taken and recorded at the location of the UAV just before the start of the flight, during takeoff.

$$x = -(x_{ecef} - x_{ref}) * \sin lon + (y_{ecef} - y_{ref}) * \cos lon [29, 30] \quad (4.7a)$$

$$y = -(x_{ecef} - x_{ref}) * \sin lat * \cos lon - (y_{ecef} - y_{ref}) * \sin lat * \sin lon + (z_{ecef} - z_{ref}) * \cos lat [29, 30] \quad (4.7b)$$

$$z = (x_{ecef} - x_{ref}) * \cos lat * \cos lon + (y_{ecef} - y_{ref}) * \sin lat * \sin lon + (z_{ecef} - z_{ref}) * \sin lat [29, 30] \quad (4.7c)$$

The user predefines the position data that the UAV should go to, but before going into the control loop as the reference data of the position controller, a reference signal is created so that it goes from 0 to the desired point linearly. The generated signal enters the position control loop as the reference signal. The difference between the reference signal and the incoming sensor data enters the proportional, integral, and derivative controller to create the roll and pitch reference signal of the UAV, and the position data of the system is measured again with the GPS sensor to complete the feedback loop.

The most challenging part of the position controller is to eliminate the noise from the GPS sensor. Figure 4.8 show the GPS sensor data collected at a fixed point for 5 minutes and converted to the NED coordinate system. As seen from this graph, the data from the GPS sensor slowly drifts up to ± 5 meters. The drift is entirely due to the GPS sensor and cannot be filtered because of the low

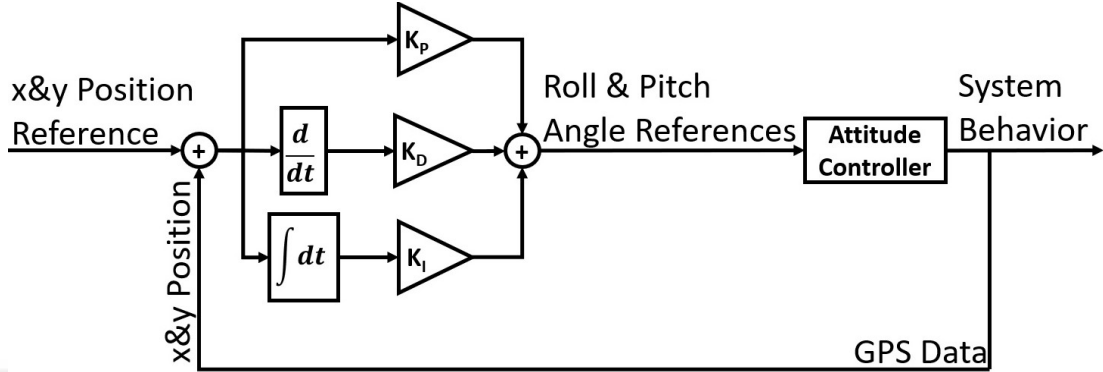
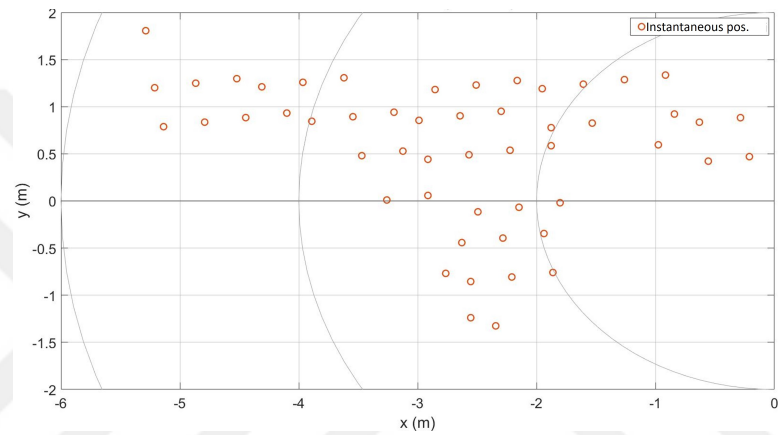


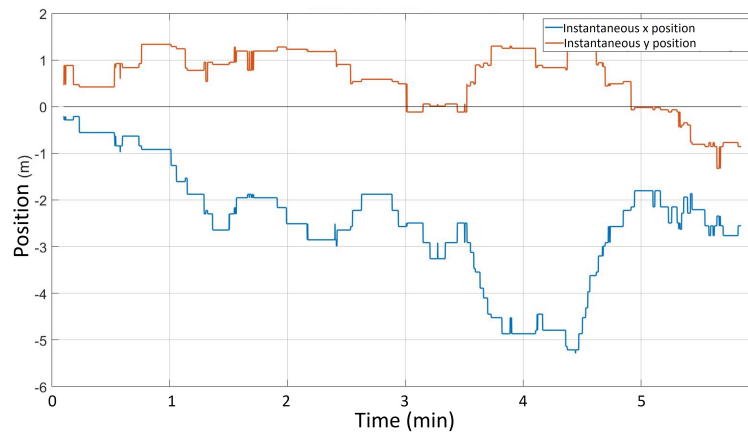
Figure 4.7: Schematics of GPS position controller

frequency of the incoming data. Therefore, during the flight, it is expected that the UAV will drift up to 5 meters while flying on the specified route.

When the experiments moved indoors, the position controller held the same, and only the feedback sensor was switched from GPS to the motion capture system. Two significant differences exist between the data received from GPS and the motion capture system. One of the differences is that the accuracy of the data from GPS is $\pm 5\text{m}$ while the accuracy of the data from the motion capture system is $\pm 0.1\text{mm}$, while the other difference is that the GPS can transmit data at a maximum of 10 Hz while the motion capture system can transmit data up to 120 Hz. Raw motion capture data is used as feedback at 120 Hz, and the outputs of the PID loops are fed to the GepRC as roll and pitch angle references, as shown in Figure 4.7. With this controller structure, the robot is able to control its speed by feeding linearly increasing (or decreasing depending on the movement direction and heading) position reference into the loop with specific increments calculated via a specified speed.



(a)



(b)

Figure 4.8: GPS data gathered for around 5 minutes while the sensor is stationary.
(a) x vs y scatter plot (b) time vs x&y

Chapter 5

Experiments and Results

5.1 Attitude Controller

As explained in detail in the Chapter 4 of this thesis, the selection of the controller constants of the attitude control loop, which is executed by the program of the ESP32 onboard, plays a vital role in the drone flying stably. A trial and error method was followed to find the values of these constants. During all the flight tests regarding the attitude controller, the robot received angular position references from the user. The initial flight tests were conducted utilizing a simple PID loop, but regardless of the different values of the constants, the drone could not sustain a stable flight and crashed during every trial. Consequently, utilizing a cascaded PID loop structure became crucial. A greater scope of the results of the experiments with a simple PID attitude controller can be seen in Appendix A. The constants of the inner loop controller are K_p and K_d for proportional and derivative terms, respectively. K_p^0 and K_i^0 are used for proportional and integral terms of the outer loop controller. The way these constants are chosen is based on trial and error. In order to find these constants, the constants of another drone, which is similar in size to our robot, were initially implemented in the program. The drone is not expected to work with these constants immediately, and the core purpose of using these constants is to create a starting point. The change

in the roll and pitch angles of the UAV with time during the flight test with the initial constants can be seen in Figure 5.1.

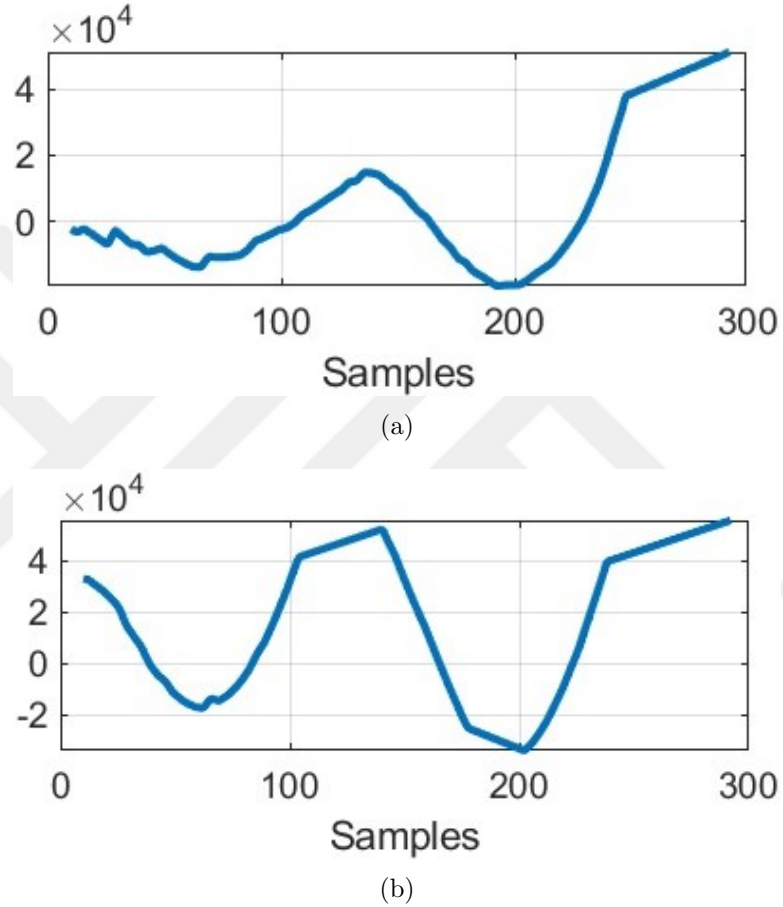


Figure 5.1: Robot's roll and pitch angles vs. sample number. The angle values are multiplied by 100 and samples are gathered in 6 seconds. (a) For roll angle, (b) For pitch angle

In Figure 5.1, the x-axis indicates the number of control loops from which the data sample was taken, while the y-axis is the pitch and roll angles in degrees multiplied by 100. As can be seen from these graphs, due to the control constants of the controller used, with each passing cycle, the error grew larger instead of getting smaller and pushed the system into instability. During the flight, considering the behavior of the drone, the constants were updated in each trial, provided that only one constant was changed at a time. These trials can be seen in Table 5.1.

Table 5.1: PID constants used in some of the experiments

Experiment	K_p^0	K_i^0	K_p	K_d
1	15.2	20	0.01	0.1
2	1	20	0.01	0.1
3	7.6	25	0.01	0
4	10	20	0.01	0.1
5	15.2	10	0.01	0.1
6	15.2	0.1	0.01	0.05
7	15.2	0.1	0.1	0.05
8	15.2	0.1	0.02	0.05
9	15.2	0.1	0.05	0.05
10	15.2	0.1	0.015	0.05
11	15.2	25	0.01	0
12	15.2	10	0.05	0
13	15.2	10	0.015	0
14	15.2	20	0.01	0
15	15.2	20	0.02	0.1
16	15.2	20	0.02	0
17	15.2	20	0.03	0.1
18	15.2	25	0.1	0
19	15.2	25	0.02	0
20	15.2	25	0.05	0
21	15.2	25	0.015	0
22	30	0.1	0.01	0.1
23	40	1	0.01	0.1
24	50	10	0.01	0.1

Table 5.1 shows some of the controller constants utilized in the flight tests conducted to date. Some of the experiments were not included in this table because the data could not be recorded or did not result in a flight worth recording. In order to find the controller constants of this drone, which both reacts as fast as desired and displays a stable flight, a total of approximately 200-250 flight tests were conducted. Based on the data obtained so far, the best performance in terms of both response time and stable flight was obtained with values of 10 for K_p^0 , 0.01 for K_i^0 , 0.0179 for K_p , and 0.05 for K_d . An extensive image of the conducted experiments is shown in Appendix A.

5.2 High-Level Controllers

5.2.1 Altitude Controller

The altitude controller of the robot receives a feedback signal from a digital barometer filtered via a one-dimensional Kalman estimator, as explained in Control and Operation chapter. The controller constants for proportional (K_p), derivative (K_d), and integral (K_i) are determined by trial and error during experimentation. The experiments for determining the K_p , K_d , and K_i constants of the altitude controller follow these steps in order:

1. K_p , K_i , and K_d constants are determined together with the desired altitude and uploaded to the robot.
2. The robot receives the takeoff command from the operator.
3. The operator controls the yaw, roll, and pitch angles of the robot via a computer. The altitude controller controls the throttle value of the GepRC.
4. After around 40 seconds, the robot receives the land command from the operator and lands.
5. The data recorded during the flight is extracted from the robot and saved to the computer.
6. The data are analyzed, and new K_p , K_i , and K_d constants are assigned.

This procedure is followed until a stable altitude controller is obtained.

Before starting the experiments to select the parameters of the altitude control loop, to avoid sudden spikes in the controller output, instead of sending the altitude reference, which is defined before the flight, to the controller as a step input, it was increased linearly to reach the desired level in 10 seconds and sent as such. The process of determining the controller parameters starts with finding

a reasonable value for the parameter K_p . The first experiments for this started by selecting the K_p parameter as 5. The results of the flight test with the K_p parameter equal to 5 are given in Figure 5.2. As seen in Figure 5.2, the oscillations in the system have become smaller and smaller and reached a steady-state equilibrium. At this equilibrium point, the steady state error is at a level that cannot be neglected. In order to eliminate the error at the steady state point, an integral term will be added to the system.

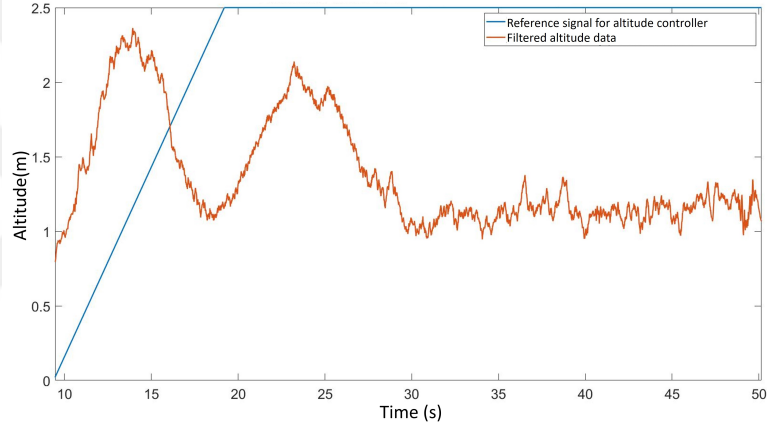
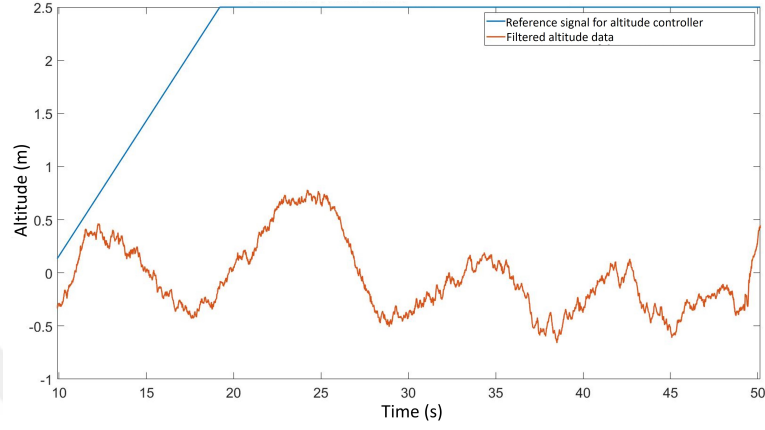
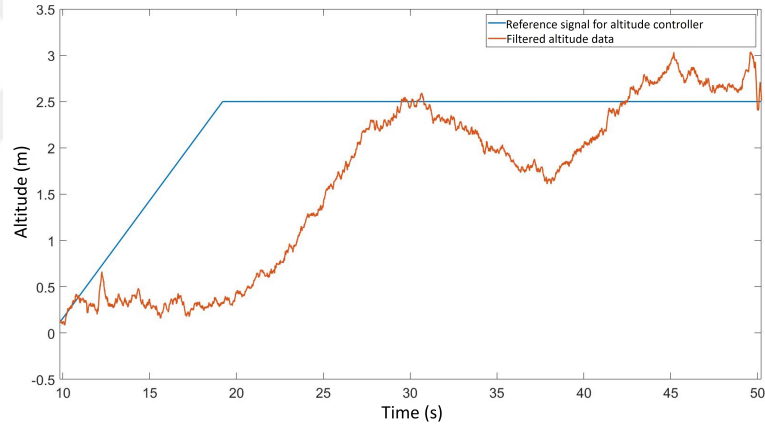


Figure 5.2: Result of the altitude flight with PID constants $K_p=5$, $K_i=0$, and $K_d=0$

Flight experiments with the addition of an integral term first started by setting the integral constant K_i equal to 0.01. The main reason was to start small and examine the effect, but no effect could be observed since it was too small. At the same time, the remaining battery percentage of the UAV also significantly affects the steady-state error and the response of the system. At this point, it was anticipated that adding an integral term would eliminate this problem. As a result of some experiments done by slowly increasing the integral term, the most successful system response was obtained when the K_i parameter was increased to 1. The result of the flight test with K_p equal to 5 and K_i equal to 1 is given in Figure 5.3(b). As can be seen from this graph, although it takes some time for the UAV to reach the desired altitude, the operating performance of the altitude controller has increased considerably.



(a)



(b)

Figure 5.3: Result of the altitude flight with PID constants (a) $K_p=5$, $K_i=0.01$, $K_d=0$, (b) $K_p=5$, $K_i=1$, $K_d=0$

Since the controller works relatively slowly with the K_p , K_i , and K_d parameters set to 5, 1, and 0, respectively, proportional constant, K_p is decided to be increased. The K_p constant increased gradually, from 5 to 35, to have a clearer image of how K_p affects the system. Figure 5.4 shows the result of a flight test with a K_p parameter of 10. As seen from this graph, the oscillations in the system are getting smaller. However, it does not work fast enough and cannot reach the desired height.

In order to speed up the response of the system and to reduce the steady-state error, the K_p parameter was slightly increased to 20. The resulting controller had

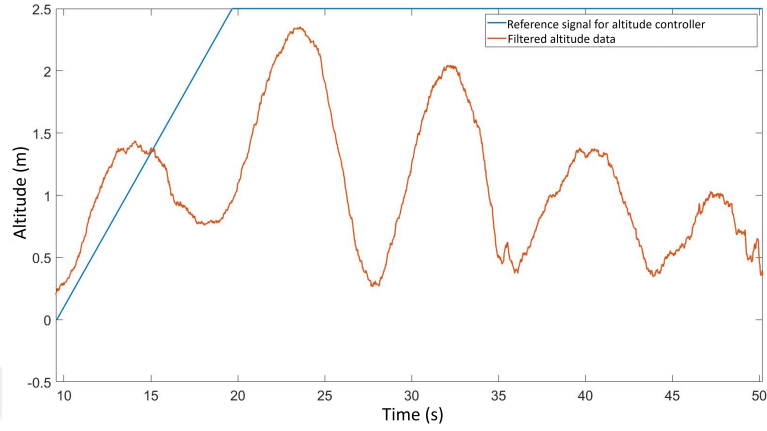


Figure 5.4: Result of the altitude flight with PID constants $K_p=10$, $K_i=0$, and $K_d=0$

a K_p of 20, K_i of 0, and K_d of 0. The graph obtained after the flight test with this altitude controller is given in Figure 5.5. As can be seen from this graph, the oscillation in the altitude of the robot neither decreases nor increases, so it does not reach to steady state. However, for the UAV's altitude control to work correctly and not to add vibration to the system, the oscillation of the altitude controller needs to be damped. For this purpose, the K_p parameter was set equal to 5, and one more flight test was performed.

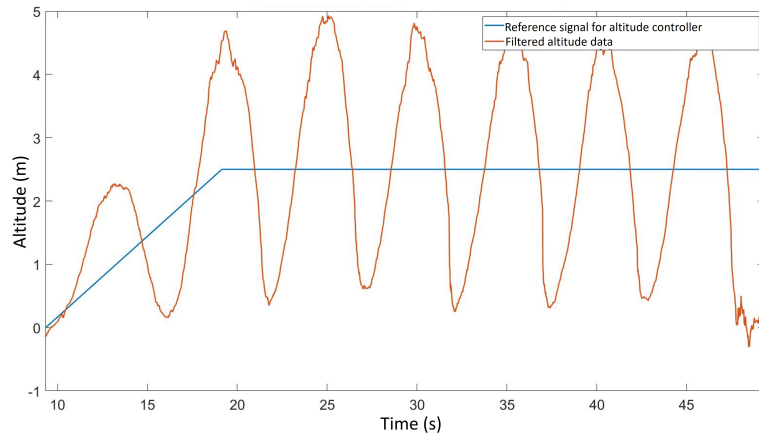


Figure 5.5: Result of the altitude flight with PID constants $K_p=20$, $K_i=0$, and $K_d=0$

A derivative term, K_d , was added to the system to reduce this oscillation. At the same time, new parameters $K_p=20$, $K_i=4$, and $K_d=10$ were chosen to

speed up the system. The results obtained from the flight test of the altitude controller with the new parameters are given in Figure 5.6. It is observed that the effect of the new integral and derivative parameters improves the performance of the altitude controller while the oscillations in the system gradually increase and become unstable.

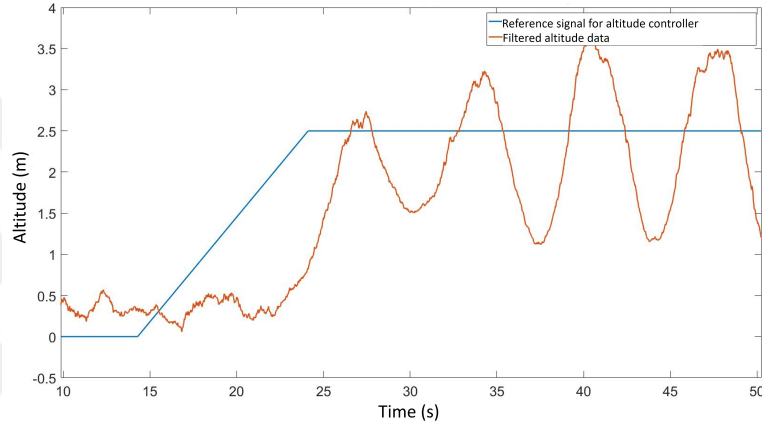


Figure 5.6: Result of the altitude flight with PID constants $K_p=20$, $K_i=4$, and $K_d=10$

As a result, the K_p value was reduced to 15, and the UAV's altitude control loop was completed. Figure 5.7 shows the results of the flight test with the final version of the altitude controller.

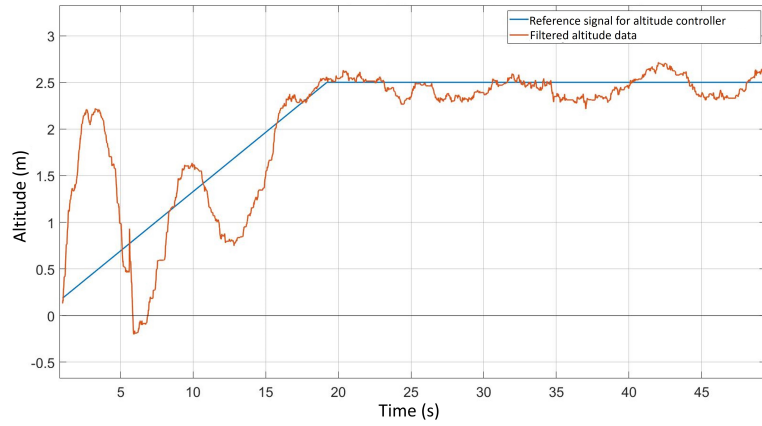
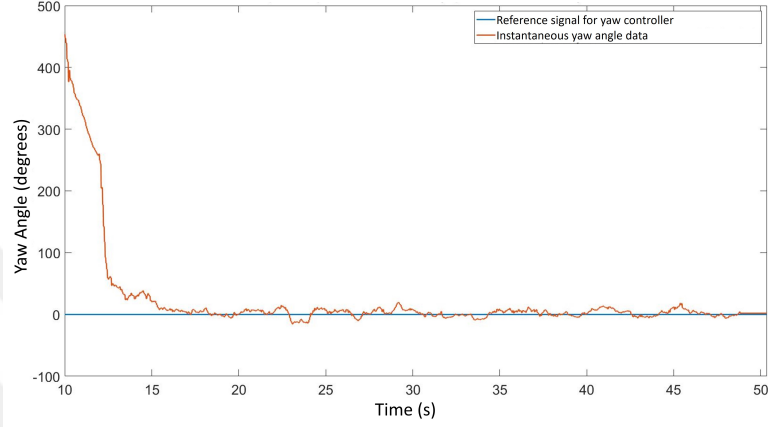
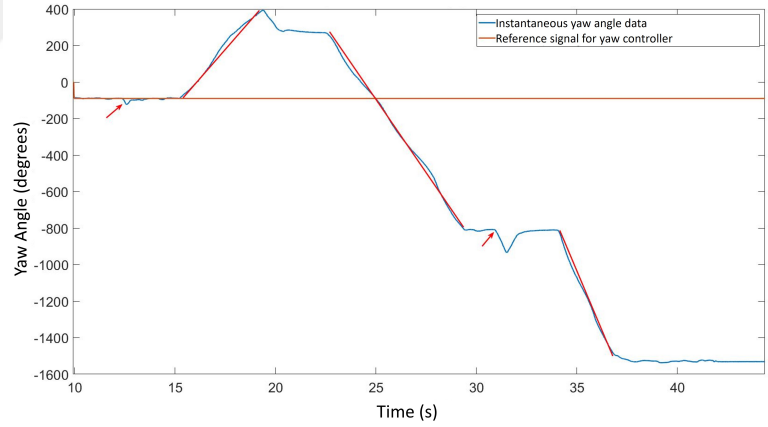


Figure 5.7: Result of the altitude flight with PID constants $K_p=15$, $K_i=4$, and $K_d=10$

5.2.2 Yaw Angle Controller



(a)



(b)

Figure 5.8: Result of the yaw controller flight. In the graph, along the red arrow and red lines, the system was disturbed from the equilibrium point by external forces. (a) $K_p=1$, $K_i=0$, $K_d=0$, (b) $K_p=5$, $K_i=0$, $K_d=0$

The yaw angle control loop is a proportional (P), integral (I), and derivative (D) controller like the altitude control loop. In order to find the required parameters K_p , K_i , and K_d , the same procedure as for the altitude controller was followed, and K_p was set to 1, K_i 0, and K_d 0. The result of the flight test with this controller is shown in Figure 5.8(a). Such a simple controller worked surprisingly well. However, as can be seen from this graph, when the reference is set to 0, the UAV returns to 0. It is sufficient to return to any angle added to

or subtracted from any multiples of 360 by wrapping the feedback signal around 0 and 360 degrees. For that, the reference values were updated, and the UAV returned to the nearest reference angle. In Figure 5.8(b), the yaw angle controller is perfected by slightly increasing K_p . In the graph, along the red arrow and red lines, the system was disturbed from the equilibrium point by external forces. As can be seen from this graph, the yaw angle controller works successfully.

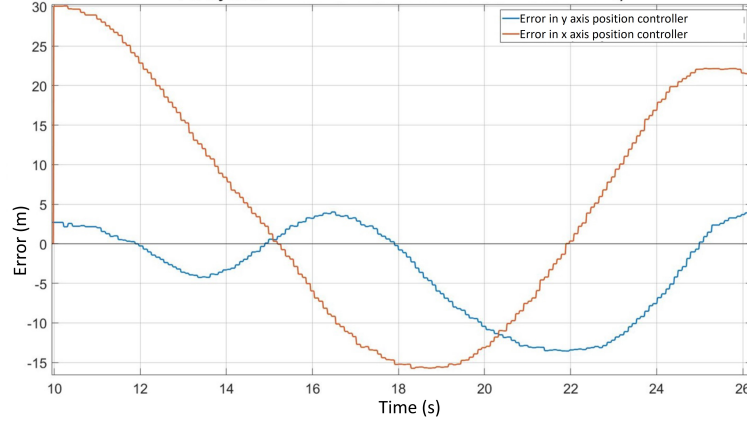
5.2.3 Position Controller

Since the GPS data contains low-frequency drift noise, the GPS data is used in its raw form. However, due to the low frequency of the GPS data, the raw GPS data causes the derivative term of the controller to give very high results. Therefore, a low-pass filter was added for the derivative term.

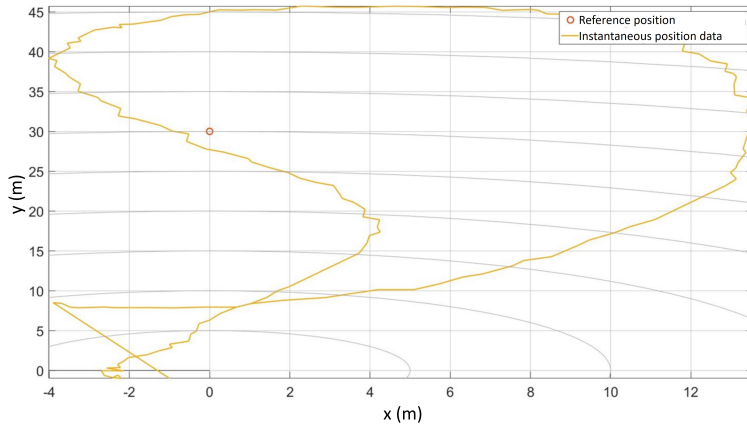
In the first flight test with the position controller, the controller parameters were selected as $K_p=70$, $K_i=0$, and $K_d=0$. The data taken during this flight are shown in Figure 5.9. As can be seen from these graphs, the UAV traveled very fast and passed the desired point during its autonomous flight. The main reason for this is that the K_p parameter is very high. The fact that the K_p parameter is too high can be seen from the gradual increase in the oscillation amplitude in the instantaneous position error on the y-axis in Figure 5.9(a).

When it was realized that a value of 70 was too high for the K_p parameter, a flight test was performed with $K_p=35$, keeping K_i and K_d constant. Figure 5.10 show the results of this flight test. As can be seen from the graphs in Figure 5.10(a), reducing the K_p parameter to 35 ensured that the amplitude of the oscillations did not increase and remained constant. This situation, which is called critically stable, can be interpreted as lowering the K_p parameter further for a stable autonomous flight.

During the testing of the controller parameters, the reference signal entering the controller was previously set as a step input. However, at the beginning of the autonomous flight, i.e., the position control loop, the system's error suddenly



(a)



(b)

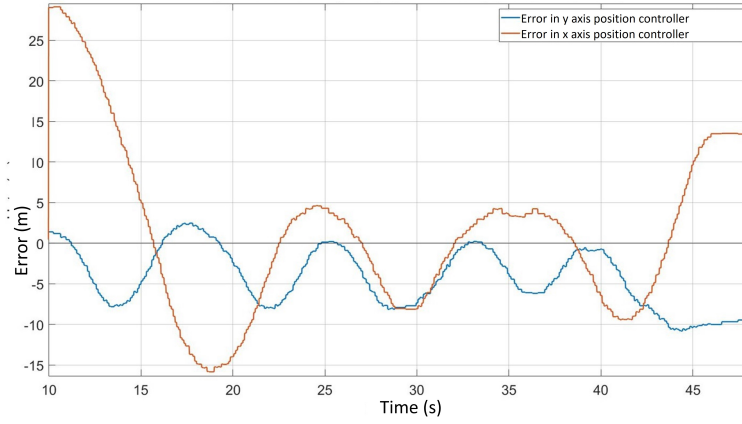
Figure 5.9: Result of the GPS controller flight with PID constants $K_p=70$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position

jumps to a very high value, moving the system away from the equilibrium state. The step input is replaced with a third-order signal that slowly increases to prevent this jump in the system error. The difference between the two signals can be clearly seen in Figure 5.11.

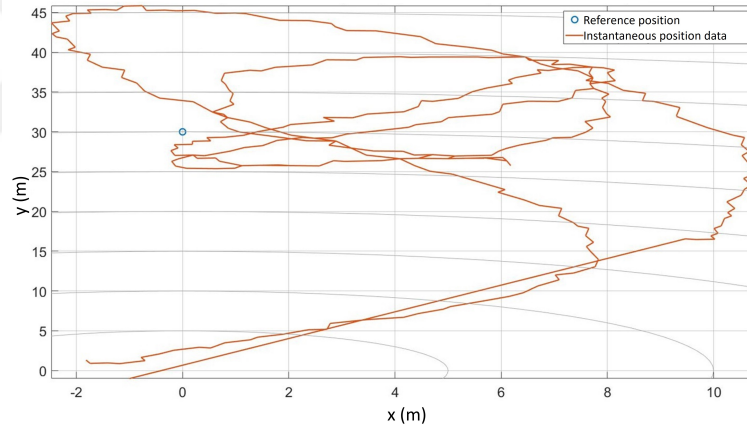
$$x_{ref} = -0.0075 * t^3 + 0.225 * t^2 \implies (y \leq 30) \quad (5.1a)$$

$$x_{ref} = 0.0075 * t^3 - 0.775 * t^2 + 18 * t \implies (y \geq 30) \quad (5.1b)$$

The mathematical equation of the reference signal of the position controller is



(a)



(b)

Figure 5.10: Result of the GPS controller flight with PID constants $K_p=35$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position

given above, and this reference signal is used in all flight trials discussed further. Figure 5.12 show the results of a flight experiment with the altered reference signal. The controller parameters were left unchanged at $K_p=35$, $K_i=0$, and $K_d=0$ to ensure that the experiments were controlled and that the cause of the changes in the result could be determined appropriately. As seen in Figure 5.12(a), although the oscillation of the system initially became smaller, there was not much difference in the oscillation of the system. It can be interpreted that the system is still critically stable, as shown in the graph of the error in the position controller of the system. For this reason, the parameters were changed, K_p was reduced to

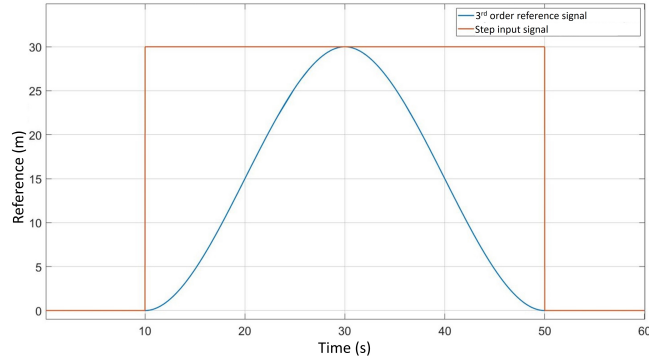
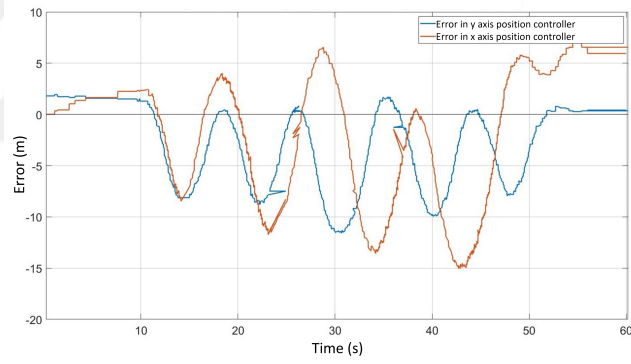
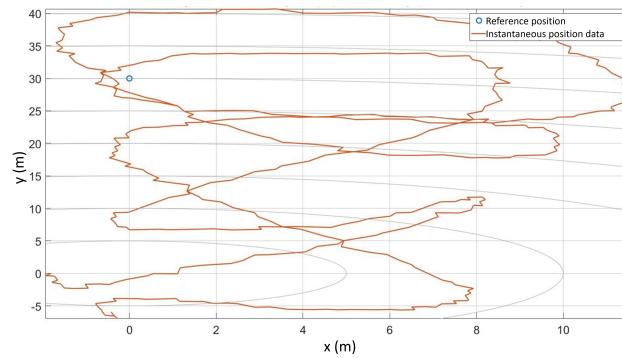


Figure 5.11: Different reference signals utilized in the position controller loop

15, and K_i and K_d were left as 0.



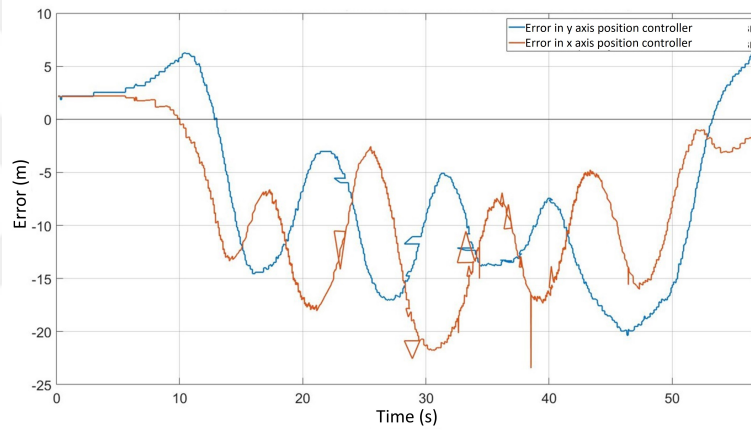
(a)



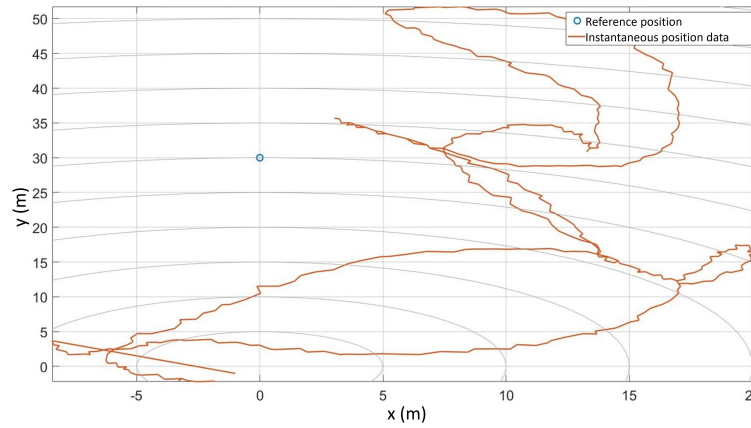
(b)

Figure 5.12: Result of the GPS controller flight with PID constants $K_p=35$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position

The error data obtained during the flight test when the K_p parameter was reduced to 15 are given in Figure 5.13(a), and the position data obtained are shown in Figure 5.13(b). As can be seen from the graph in Figure 5.13(a), the error oscillations in the position controller started to get smaller and smaller. Since the oscillations are not fully damped, the K_p parameter should be slightly lowered. Figure 5.13 shows the position data of this flight experiment, and it can be seen that the UAV exhibited a more stable flight.



(a)

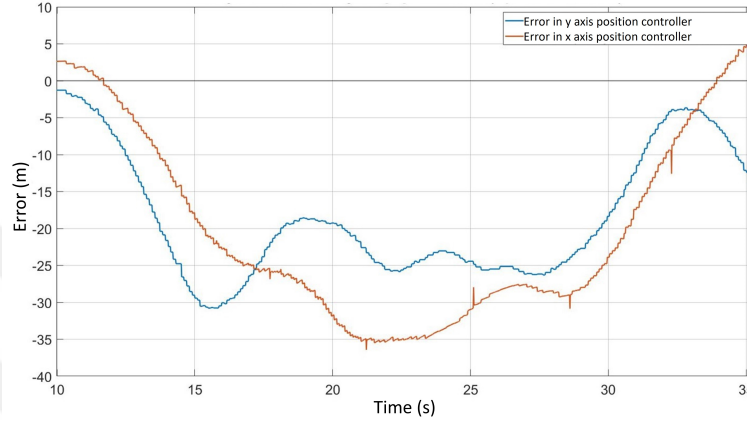


(b)

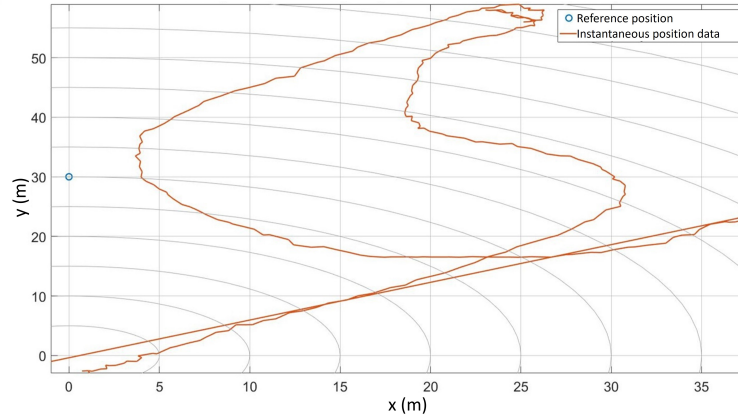
Figure 5.13: Result of the GPS controller flight with PID constants $K_p=15$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position

Since the oscillations were not damped when the K_p is 15, K_i and K_d are 0,

K_p was slightly lowered to 5. The results of the flight test with parameters $K_p=5$, $K_d=0$, and $K_i=0$ are shown in Figure 5.14.



(a)



(b)

Figure 5.14: Result of the GPS controller flight with PID constants $K_p=5$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position

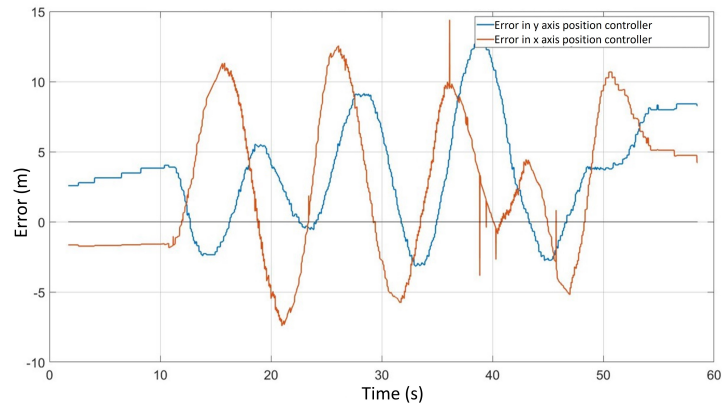
In the flight tests described so far, the signs of the position controller's output are always set so that the front of the UAV always faces north, i.e., the UAV must pitch positively to move north and negatively to move south. Similarly, the UAV needs to roll positively to head east and negatively to head west. However, the camera mounted on the UAV must always face the direction in which the UAV is moving. Therefore, the UAV's position controller pattern was changed so that

it would first turn in the direction of its destination and then perform a positive pitch in that direction. Since such a fundamental change was made to the position controller, it is necessary to re-examine the controller parameters experimentally. During the flight trials, it was seen that K_p alone was not sufficient, so K_i and K_d parameters should be added to the position controller. However, the experiments were conducted from scratch since the working principle of the UAV's position controller was changed before the K_i and K_d parameters were altered.

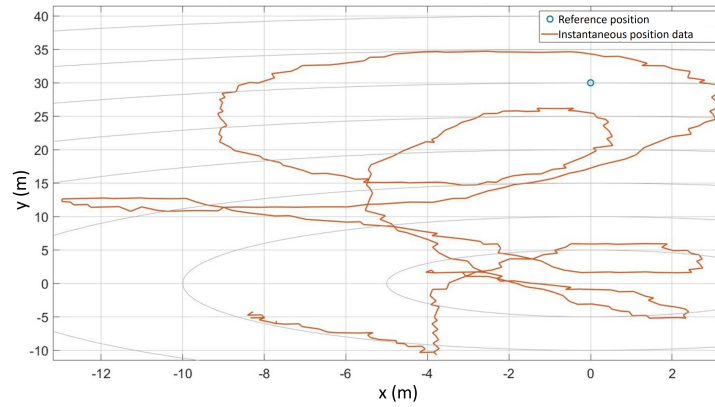
To examine the changes made to the drone's position controller more extensively, the body reference system and the reference system that takes the direction of movement as the positive x-direction, which both were previously completely independent from each other, now coincide. In other words, the direction of movement of the UAV is the same as the direction it is facing, i.e., the direction where the camera is facing. In order to overlap these two reference systems, a yaw angle controller is added to the position controller, and the reference signal of the yaw angle controller is calculated from the error of the position controller. Therefore, the robot always turns towards its destination first, then pitches forward. The mathematical equation used to obtain the reference values of the yaw angle controller using the position controller data is given below. Since the "atan2" function is used in the program as the inverse of the tangent function, there is no need to decide in which part of the coordinate system the obtained angle value is located.

$$yaw_{ref} = \tan^{-1} \left(\frac{y_{ref} - y_{current}}{x_{ref} - x_{current}} \right) \quad (5.2)$$

The experiments using the new position controller model started with a K_p parameter of 30, K_i , and K_d of 0. However, no different results were obtained from the flight experiment with the previous position controller with $K_p=35$. As can be seen from the error graph in Figure 5.15(a), the oscillations in the system need to be damped by decreasing the K_p parameter. Figure 5.15(b) shows the path of the UAV during this flight test.



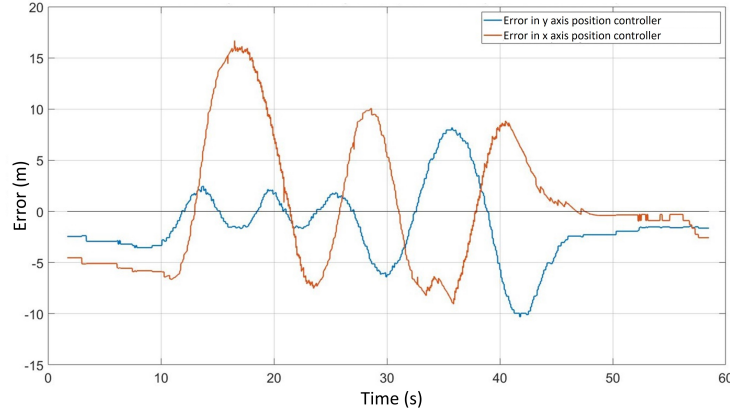
(a)



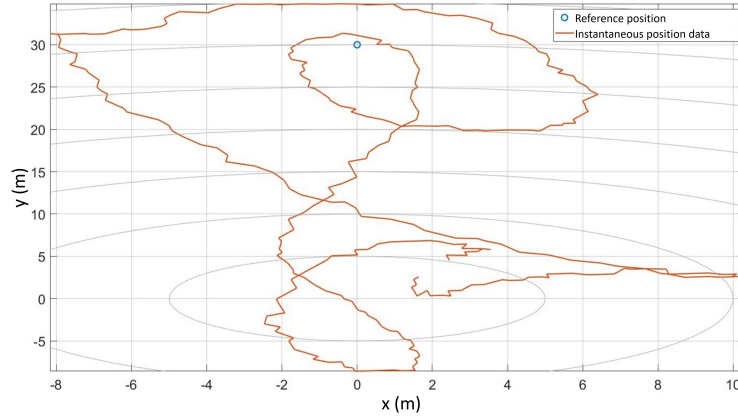
(b)

Figure 5.15: Result of the updated GPS controller flight with PID constants $K_p=30$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position

The K_p parameter was reduced to 25. While K_d and K_i parameters are kept at 0, as seen from the graph in Figure 5.16(a), a slow damping is observed, especially in error on the x-axis. Since this damping is very slow, the K_d parameter was added to the system to speed it up. The path of the robot in this experiment is given in Figure 5.16(b).



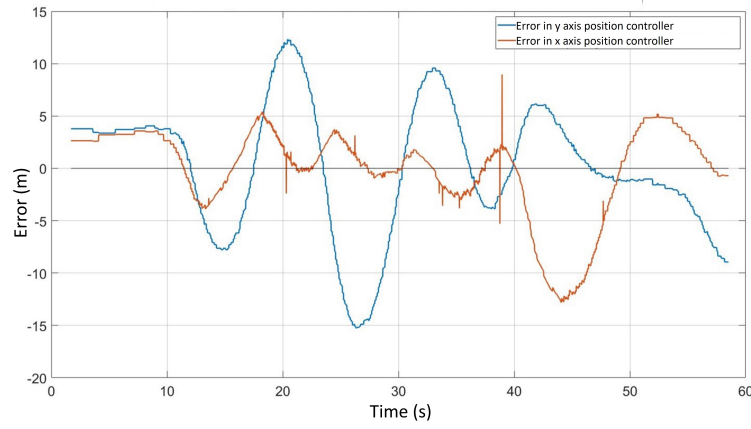
(a)



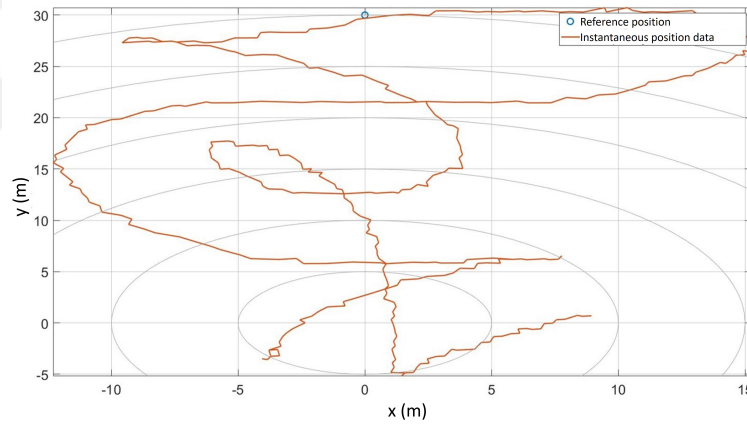
(b)

Figure 5.16: Result of the updated GPS controller flight with PID constants $K_p=25$, $K_i=0$, $K_d=0$. (a) Error vs. time, (b) x position vs. y position

The K_d parameter was increased from 0 to 0.1, and the K_p parameter was increased to 35. Figure 5.17(a) shows the position controller error data from a flight test with the new parameters. Adding the K_d parameter to the position controller resulted in a noticeable reduction of the error in the x-axis. In the y-axis, it is observed that the oscillation is gradually damped. Figure 5.17(b) shows the path of the UAV during the flight.



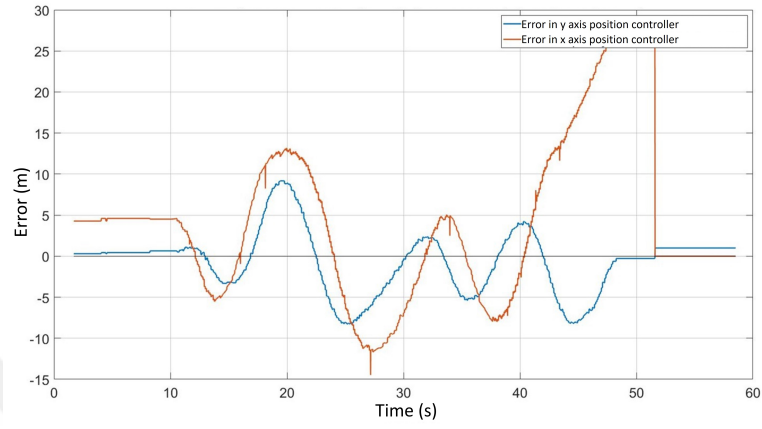
(a)



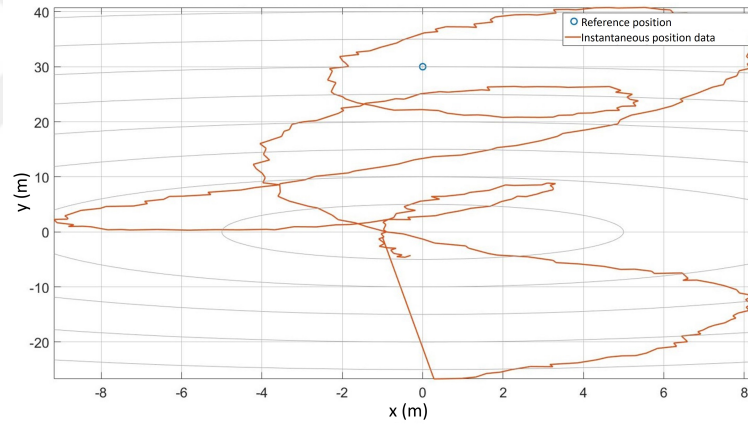
(b)

Figure 5.17: Result of the updated GPS controller flight with PID constants $K_p=35$, $K_i=0$, $K_d=0.1$. (a) Error vs. time, (b) x position vs. y position

In order to reduce the oscillations created by the position controller in the system, the K_d parameter was further increased and set to 1. According to the graph in Figure 5.18(a), the oscillations in the system tend to decrease gradually, albeit slowly. Figure 5.18(b) shows the flight path of the robot. Apart from the flight experiments described, a few other experiments were conducted, but the results of these experiments made the position controller even more unstable. These experiments are with constants " $K_p=30$, $K_i=0$, $K_d=0.1$ ", " $K_p=40$, $K_i=0$, $K_d=0.1$ " and " $K_p=25$, $K_i=0$, $K_d=1$ " respectively.



(a)

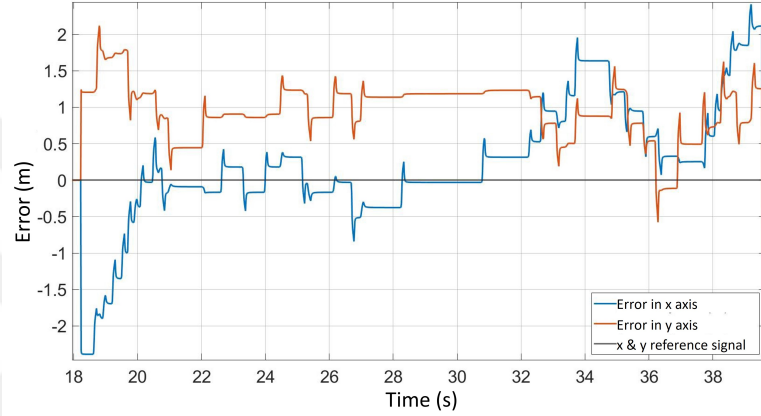


(b)

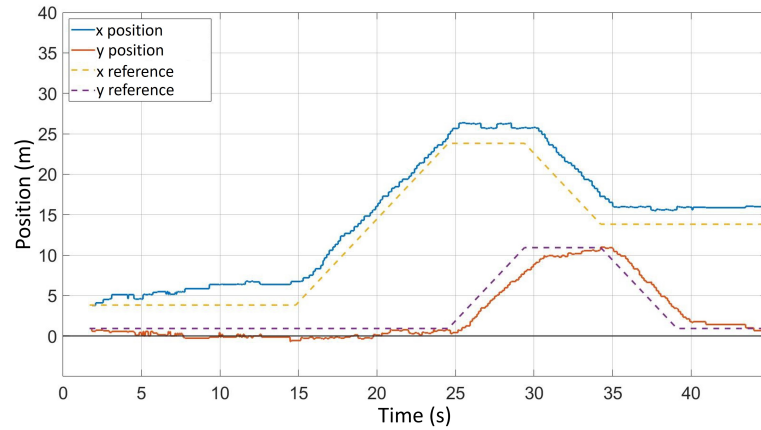
Figure 5.18: Result of the updated GPS controller flight with PID constants $K_p=35$, $K_i=0$, $K_d=1$. (a) Error vs. time, (b) x position vs. y position

As seen clearly from the abovementioned Figure 5.18, the position controller does not work stably and has a tremendous error. Thus, there was no noticeable improvement even though many different controller constants were tested. Because of this, the filtering method was changed completely. Since the data frequency is too low, the low pass filter, which was used, adds a non-negligible delay to the controller, which makes the whole system unstable. However, due to the low frequency of the GPS data, the raw GPS data causes the derivative term of the controller to give very high results. Therefore, a low-pass filter was

added for the derivative term. Removing the low pass filter, using the raw data as the feedback signal for the position controller, and adding a low pass filter to the output of the derivative term of the controller, improved the performance and stability drastically. Stable outdoor flight data is shown in Figure 5.19.



(a)

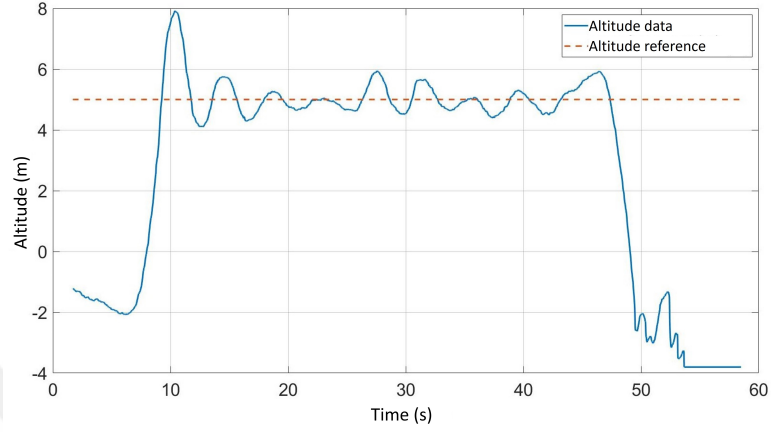


(b)

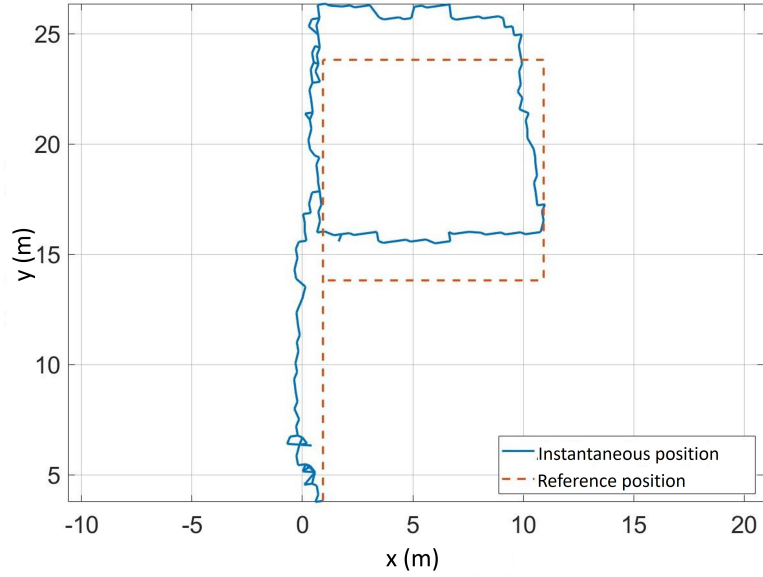
Figure 5.19: (a) The results of the experiment using exactly the same controller and filter structure immediately after the indoor flight tests, (b) The results of the outdoor flight after the mentioned adjustments and finetuning of the controller constants were made

The robot's position controller, altitude controller, and yaw angle controller were successfully executed to create an autonomous flying robot. Finally, the

robot was able to take off and land on its own. Autonomous takeoff was achieved by linearly increasing the nominal throttle value, which is also used in indoor flights, and after the UAV reached a certain altitude, the altitude controller was activated. The calculated nominal throttle value was added to its outputs. Autonomous landing is achieved by entering the nominal throttle value as the throttle signal of GpRC and decreasing this value linearly. As shown in Figure 5.20, autonomous landing and takeoff take approximately the same amount of time. When the position controller, altitude controller, and yaw angle controller are combined together, a fully autonomous drone is obtained. As seen in Figure 5.20, the UAV is able to fly on its own following a square path which is a predetermined trajectory.



(a)



(b)

Figure 5.20: (a) The performance of the robot's altitude controller with autonomous take-off and landing, (b) The path following performance of the fully autonomous robot with autonomous take-off and landing

Moreover, to show the fully autonomous flight capabilities of the robot, a few extra path-following tests are done. Before every test flight, if the operator is an experienced pilot, they first determine the maximum speed at which the UAV can take off, depending on the weather and wind speed; if not, the robot chooses

a predetermined safe speed. Once the speed is determined, the target altitude of the robot with respect to the takeoff point is specified. Then, the coordinates of the positions where the robot needs to go are entered into the program, and the robot is initialized for flight. After these values are defined, the robot is placed on the ground in an open area suitable for takeoff. The robot, which needs around thirty seconds to initialize the sensors, starts its defined task with the "takeoff" command from the operator, and when the defined mission is finished, the robot returns to the point from which it took off and lands.

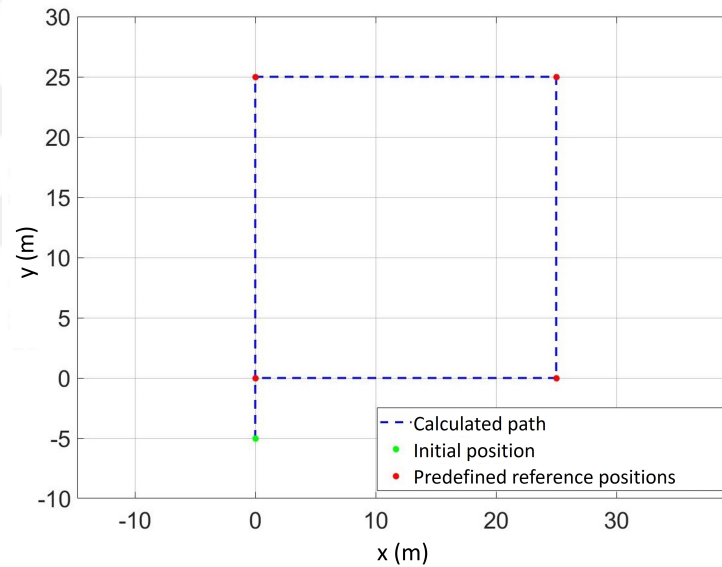


Figure 5.21: Demonstrates the operation of the flight planner and path generation

In addition to the controllers, a flight planner algorithm is required for the drone to fly autonomously in a stable manner. This algorithm creates the path that the drone will follow, not only to get the drone to the location specified by the operator but also to ensure that the drone flies at the specified speed. To do this, the flight planner only needs the coordinates of where the drone needs to go and the speed at which the drone will fly. The flight planner first determines the number of points that the operator enters into the system. It then divides the flight into segments, with one segment between each two consecutive points. The flight planner takes the first point where the drone is located and creates a route between that point and the first point, which the user defines. This route is, in fact, a series of position indexes consisting of the coordinates that

the drone should follow step by step. Since the cycle time is fixed at 5 ms, the flight planner calculates the coordinates where the drone should be at the next step according to the defined speed. The position at the next step is found by adding the distance the drone has to cover in 5 ms to the current position. At the same time, the flight planner monitors whether the drone has reached the position defined by the operator and decides whether or not the drone should move to the next position. This cycle is repeated until the drone reaches all the user-defined points in sequence and finally returns to the initial point. If specified before the flight, the flight planner can also enable the drone to hold its position at a given coordinate for a certain period. Figure 5.21 shows a square path generated by the flight planner.

When the controllers and the flight planner are utilized together, the robot only waits for a command from the operator, which can be sent with a single button. As soon as the robot is connected to the battery, it gets its current position with the help of GPS, and all the coordinates that the robot needs to navigate are recalculated with respect to this starting point. After the pre-flight checks and calibrations are finalized and the robot's sensors are initialized, the robot rotates its motors at the lowest speed, signaling the operator that it is ready. According to the command from the operator, the robot starts its autonomous takeoff. After reaching the specified altitude, the robot begins its mission, again with the signal from the operator. Using the same button, the operator can stop the robot in the position it is in during the mission and resume it from where it left off. After completing the task, the robot autonomously returns to its starting point, lands, and turns off its motors. In case of an emergency, to prevent any damage to life or property, it is also possible for the operator to turn off the robot's motors with a different button and let the robot fall freely.

Another result to be demonstrated in the project is that the designed robots can be grouped in different numbers and fly. The swarm flight tests started with a single robot holding its position. Then, the robot's performance in following a defined path was analyzed with trials in which the robot flew from the takeoff point to another point determined by the operator and back. Finally, the single robot flights were completed by having the robot follow a square path with four

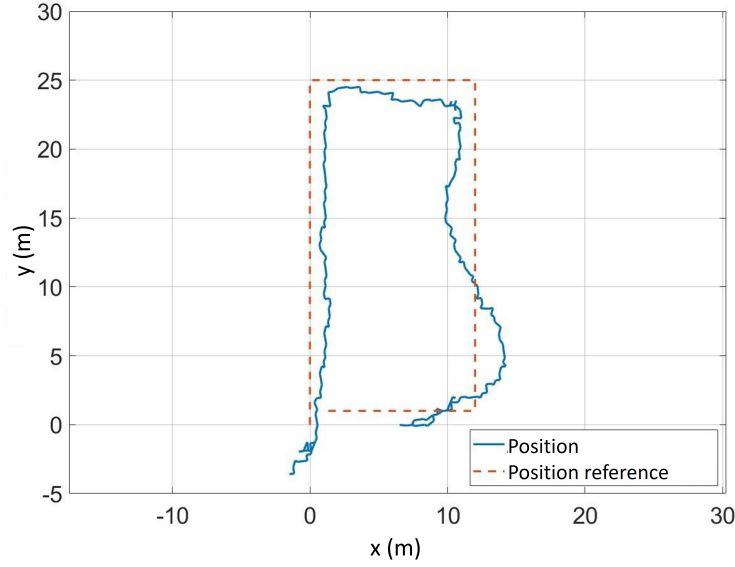


Figure 5.22: Ability of a single robot to follow the created path

corners defined by the operator. Figure 5.22 shows the path of the single robot and its performance in following it.

5.3 Sensorized Collision Resilient Aerial Robot (SCoReR)

For detection, the soft sensors attached to the bumpers are used. The raw (unfiltered) data obtained from the sensors is processed live. After the drone takes off, the responses of the soft sensors are recorded for a second to determine the average of the soft sensors' responses in the hovering position of the drone. This process has to be done in hovering conditions to compensate for the noise caused by the airflow and the vibration caused by the motors. Adding a predefined value to the average responses sets the threshold for obstacle detection. The detection algorithm checks whether soft sensors exceed the threshold in each control loop. If the threshold is exceeded, the robot understands that it has collided with an obstacle.

In Figure 5.23, all the steps of the detection procedure are shown. First, SCoReR hovers, and the variation of the soft sensor data is relatively low. Soft sensor output increases above the threshold value at the impact, which can be seen in the red-labeled regions shown in Figure 5.23. When the drone detects such a sudden increase in the output of any of the soft sensors located at the front bumpers, the drone is commanded either to turn back to its initial position or to wait at the contact position.

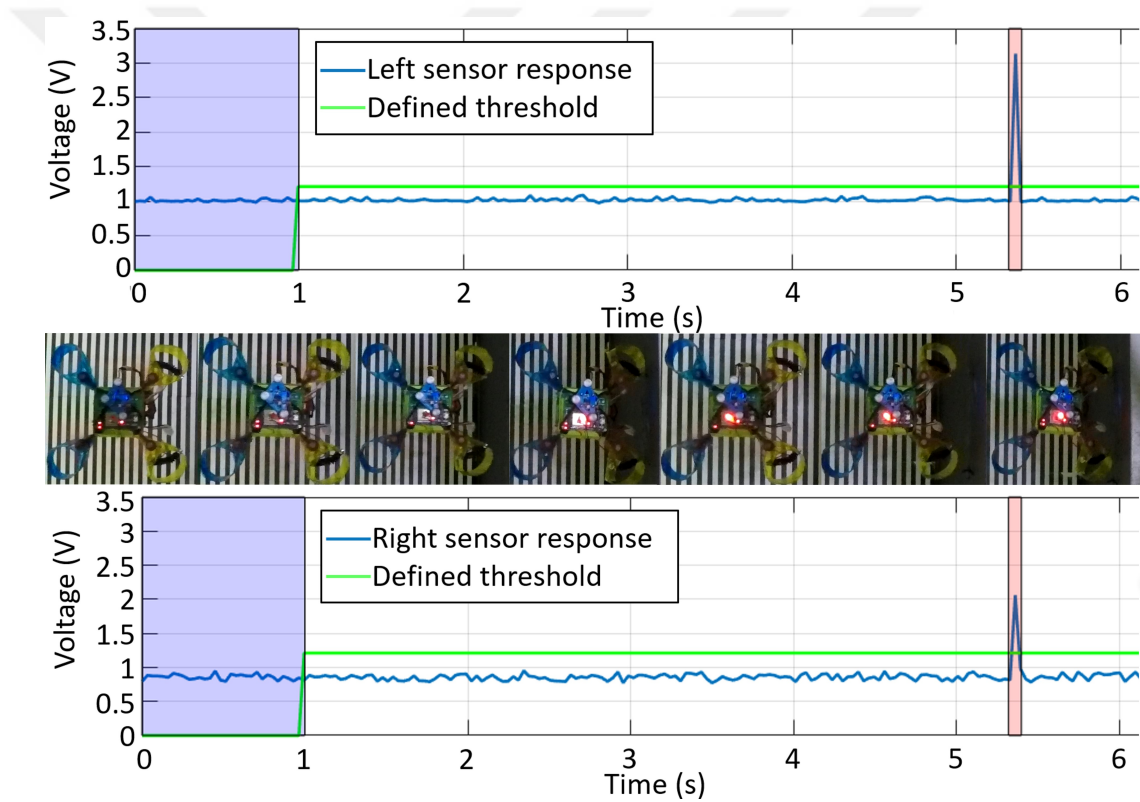


Figure 5.23: Step-by-step demonstration of the obstacle detection system

To test the functionality of the obstacle detection algorithm, we flew the drone toward the wall at different speeds, starting from 0.1 m/s up to 1 m/s with 0.1 m/s increments. Five trials were conducted for each speed. Soft sensors managed to detect the impact in all the trials. Therefore, the accuracy of the detection is 100%.

However, at very low speeds, such as 0.1 m/s, it takes more time for the drone to detect the wall. Sometimes the impact cannot exceed the threshold, and the

drone continues towards the wall until the soft sensor output, which is directly correlated to the contact force, exceeds the threshold. Touching the wall until the sensor detects it is not a big problem for SCoReR, as its compliant nature ensures that the robot can easily sustain such low-speed impacts.

We conducted impact tests with different speeds to show the potential of the sensor embedded in SCoReR. The experiment setup includes a black rigid wall for SCoReR to crash, a striped plate to be able to analyze the impact with a camera, a high-speed camera, and the SCoReR itself. The rigid black wall is placed so that the camera would see the impact from above. The striped plate is placed underneath the flight path.

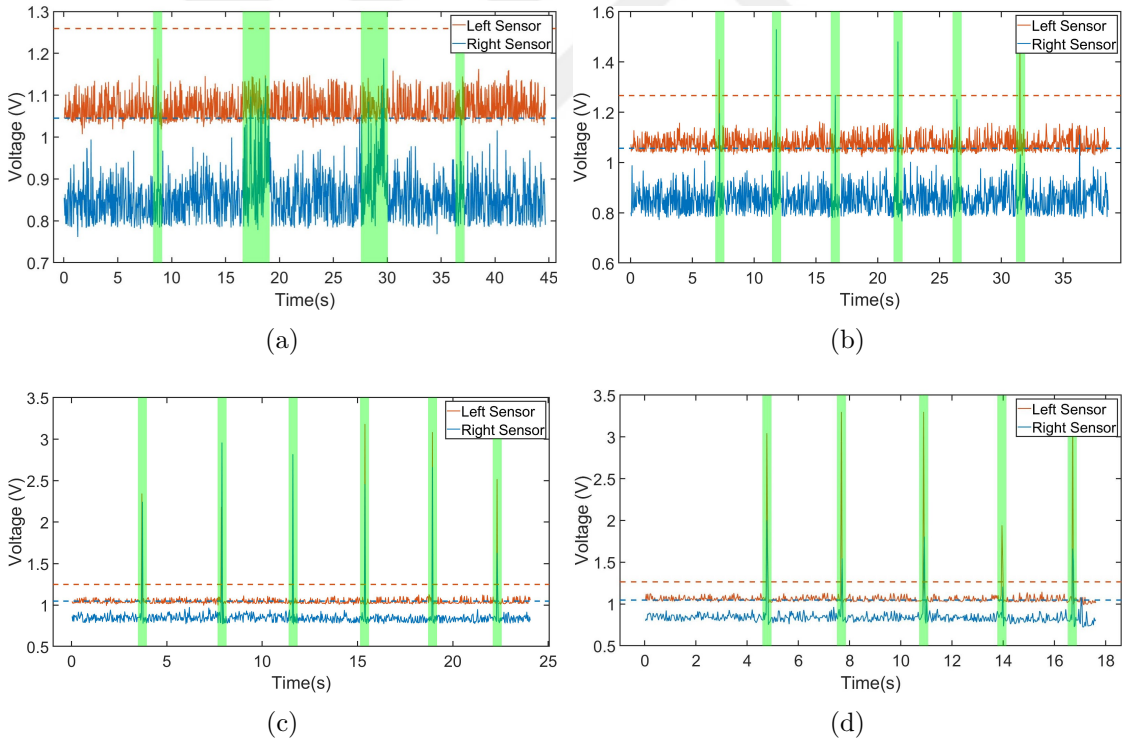


Figure 5.24: The results of sensor readings during the test flights with different speeds. Dashed lines represent the thresholds for corresponding sensors. (a) 0.1 m/s, (b) 0.2 m/s, (c) 0.5 m/s, (d) 0.9 m/s

Figure 5.24 shows the sensor voltage readings during some of the flight tests with controlled crashes. Green segments of the plots are the regions where SCoReR's sensors detect the wall. SCoReR, arms itself, gets readings from the

sensors for one second and takes their average at the beginning of the test flights. After that, SCoReR initializes the takeoff sequence until it reaches the designated altitude while holding its position on the initial point. When the takeoff sequence is finalized, SCoReR waits for a command from the user to start the impact tests. The user sends the command to initiate the impact tests, and SCoReR starts to move toward the wall with a predefined velocity. When SCoReR touches the wall, it detects contact and returns to the initial position. At this point, SCoReR is ready for a new command, either "do one more impact test" or "land". All the tests are done at least five times following the same procedure. In all of the experiments, SCoReR can detect the wall successfully. From the plots in Figure 5.24, we can observe the collision moments via the instant jumps in the data, which are represented by the green regions. After some initial experiments, we observed that a threshold of 0.2 V would work as the data during impact would exceed the threshold while the noise of the sensor output caused by robot vibrations and the airflow stays below this threshold.

Following the impact experiments, a case study is performed. During the case study, after detecting the impact via the sensor, SCoReR updates its position references and holds its position at the edge of the wall.

The main difficulty during the tests was detecting the impact during low-speed flights. The bumper absorbed most of the impact in lower-speed flights, but the deflection was too slight. Nevertheless, thanks to the sensor design, the robot can detect 0.2 m/s impacts almost instantaneously. The experiments with 0.1 m/s show that even if the sensor cannot detect the impact instantaneously, SCoReR can detect the wall after a few seconds. SCoReR tries harder to move forward during these trials and applies extra force until a sensor detects the wall.

5.4 Effect of Compliance in Drone Body

Together with the electronics and controllers, the robot has the capability of flying autonomously; however, more is needed to perform experiments to test

robot before the flight.

2. We place the robot under motion capture and define the motion capture markers on the robot as a rigid body in Motive (the motion capture system's software).
3. We plug the robot's battery, and it engages the motors.
4. We command the drone to lift off and go to the initial position, which is predefined to be located 2 meters away from the rigid structure that the drone will hit.
5. Our second command starts the experiment, and the drone flies toward the rigid structure.
6. After it hits, we command the drone to return to its initial position.
7. After reaching the initial position, we command the robot to hit again. The fifth and sixth steps are repeated at least five times to ensure that the survivability of the drone is not a one-time event but that it can actually withstand the impact with the desired speed.
8. Finally, we command the robot to land and disconnect from the battery for data extraction.

In order to calculate the position of the robot, the motion capture system, OptiTrack, must be calibrated in two stages. The first one is to collect random data points with its unique wand, which has three reflective markers with specific separation from each other. During this stage, collecting an equal number of data points, which cover the field of view (FoV) of every camera, is essential. After this stage is completed, Motive software gives an average estimation of positional error per marker, which we aimed to keep around 0.05 mm. The following step is to define a ground plane together with an origin point. OptiTrack has a rigid triangle with markers on the corners, and the side of the triangle is defined in the software. So, when this is placed under the FoV of every camera, the software can define X direction, Y direction, origin point, and an XY ground plane. For

the motion capture system to give position data of a rigid body (in this case, the robot) together with orientation data, it is needed to place at least three reflective markers on the robot so that infrared cameras can detect them as points and define a rigid body having its own coordinate system. The origin of this coordinate system is located on the center of gravity (CoG) position and orientation with respect to the global origin point defined during the calibration process. Using Motive and the sample Python program provided by the developers, we can extract position and orientation data from Motive and store it in an array in the Python program. The whole data transfer between the Python program and Motive is executed in a parallel thread instead of the main loop so that data transfer does not affect the rest of the program.

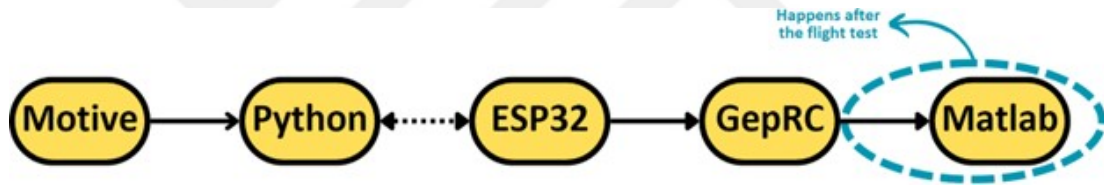


Figure 5.26: Data exchange between subsystems (The direction of the arrows indicates the direction of data transfer. Dotted arrows indicate wireless connectivity.)

The most challenging part of the data exchange between robot subsystems is the connection between the PC and the ESP32 on the drone. The wireless connection between these is established by ESP-NOW protocol, a wireless communication protocol specifically designed to connect multiple ESP32 boards in a very stable and low-powered way with a high-range peer-to-peer connection [31]. In order to utilize ESP-NOW, we connected an extra ESP32 to the PC, which acts as a ground station exchanging data with the PC and the ESP32 on the robot. In the main loop of the Python program, the data from Motive is sent to the ESP32 together with user inputs via a UART connection. The ESP32, connected to the PC, encodes a message which includes the position data of the robot measured by motion capture and the user inputs, sending the message to the ESP32 on the robot. After receiving the message, ESP32 on the robot decodes it and assigns them to their corresponding global variables, which are used in the different controller loops.

The program running inside ESP32 on the robot outputs the signals to be sent to GepRC via a UART protocol named PPM which encodes different values for up to 8 channels. The PPM signal is a combined digital signal consisting of the output of every PID loop and user inputs whenever necessary. PPM is an abbreviation for Pulse-Position Modulation, which measures the time difference between two falling edges of a signal. The time difference defines the value for a specific channel and afterward reads the next channel until all the channels are read. In short, the ESP32 onboard encodes the data into a PPM signal, and GepRC decodes and uses it as input.

The final connection is arguably the most crucial one, as it makes data gathering between experiments possible by extracting the saved data from the ESP32 and transferring it to the PC. After every test flight, we record the raw motion capture for further examining and understanding the flight behavior as well as the position and heading data recorded on ESP32 and the high-speed video footage of the flight. Storing the video and the raw OptiTrack data is straightforward; however, storing the flight data saved in the drone is relatively challenging as simultaneously saving data and sustaining flight is a resource-heavy task. Saving the data gathered in a single loop takes more time than the loop itself, so when they add up, it accumulates very fast and disrupts the PID loop stability and execution time. We solved this issue by storing the flight data inside a matrix and saving it as a text file inside the SPI Flash File Storage (SPIFFS). Even though SPIFFS is the only storage onboard the ESP32 and is used for the code during regular operation, we divided it into two parts: one for the code and the other for data storage. Because the maximum number of bytes a single matrix can occupy is limited, we had to store one data point for every eight. After the save is successful, we extract the data by uploading a different program to the ESP32, which reads the text file and prints the data points on the serial port. At the same time, on PC, we read the serial port on MATLAB and store every data point into a matrix and plot them.

In summary, autonomous test flights require several different subsystems which communicate with each other for a successful experiment. Motive software reads the position and orientation data of the drone and sends it to the Python program.

Python program encodes the data from Motive with user inputs and sends them to the ESP32 connected to PC. That ESP32 then sends the encoded data to the drone via ESP-NOW protocol. GepRC receives inputs from ESP32 on the robot as the outputs of the higher-level controller loops and inputs of the attitude controllers. Finally, the recorded data is extracted by MATLAB and stored as a `cls` file, MATLAB figure, and a video.

In the flight tests, the robot autonomously hits the rigid wall consecutively at least five times for every impact speed until it reaches the speed where the drone crashes. After any of the impacts, if the drone cannot sustain flight, the experiment is repeated, no matter if it is the first one or the fifth. If the second flight test with the same desired impact speed causes the drone to crash to the ground before the fifth hit, the reliable maximum impact speed for that body design is 0.1 m/s less than the speed where crashes start to occur. In short, if a drone cannot survive a crash at a specified speed five consecutive times, the robot can withstand collisions below that speed.

During the experiments, it was impossible to create continuous relation between the impact speed and the robot's survivability because of two main reasons; surviving a crash not only depends on the impact speed but also it depends on the impact angle, and the speed controller has some error, so the robot does not hit the rigid wall precisely at the desired impact speed. The robot starts to move as far from the wall as possible to minimize the effects of impact angle, aiming to reduce the acceleration during impact to zero, which results in a more stable and controlled collision. We also tried to reduce the controller error by changing controller constants, and the error is under ± 0.1 m/s for most of the flight tests, and for more than half of the tests, the error is around ± 0.05 m/s. Furthermore, the increments of desired impact speeds of different flight tests are set to 0.1 m/s. Because of the error and 0.1 m/s increment, the whole experiment covers the impact speeds ranging from 0.1 m/s up to the point where the robot fails without discontinuity.

The first experiment was to fly a robot with an utterly rigid body together with bumpers. We used PLA material for the rigid body since we use 3D-printed

bodies and bumpers so that we can design the parts as close as possible to their compliant counterparts. Then the experiment is done using the PLA body to test the survivability of the robot when it is completely rigid. As can be seen from Figure 5.27, the body (arms and bumpers) does not flex, and as a result, the robot does not have room for recovery and falls. Because the robot is rigid, there is close to no impact damping, and all the force generated during impact is transferred into torque to pitch the robot toward the wall. Because of the torque, the front of the robot faced directly downward, and it fell to the ground.

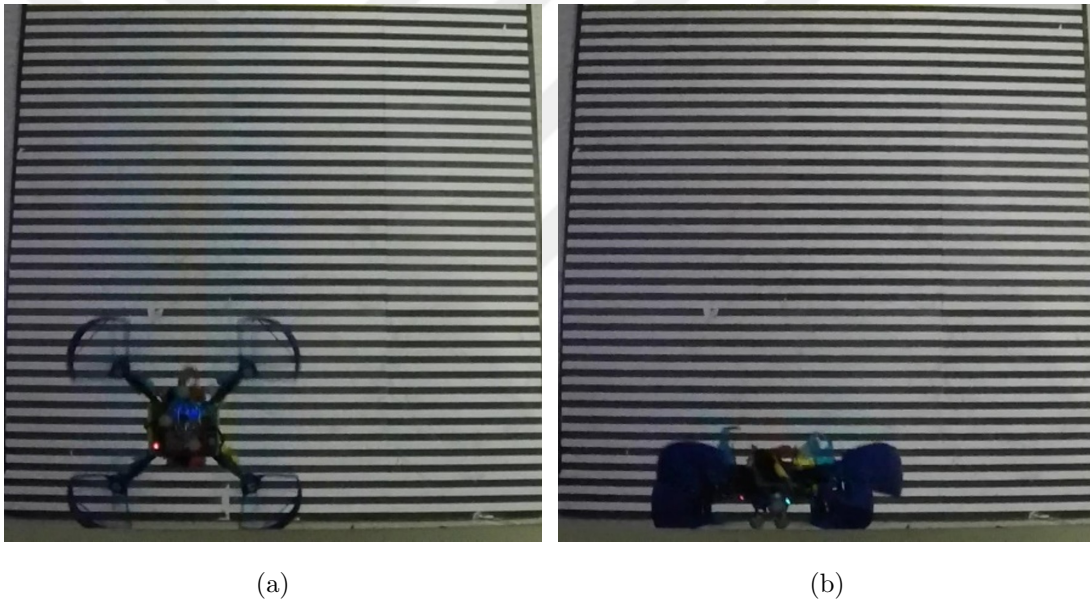


Figure 5.27: (a) Robot's impact moment. As can be seen it does not flex and does not dampen the impact. (b) Robot flips toward the wall and the front directly facing downwards.

When we switched to a completely foldable design to introduce compliance, the failure characteristics of the robot changed utterly. There is a slight increase in the maximum impact speed from which the robot will survive the crash, thanks to compliance. However, we introduced another failure mode to the system by introducing rigidity. When the robot hits a rigid wall, if its speed is around 1.9 m/s, the bumpers flex, and the body itself and propellers hit a wall. When propellers hit the wall, they immediately stop even if they are not broken, and this causes a nonzero net torque on the robot as the front propellers come to an

immediate stop. In contrast, the back ones continue their regular operation. Even though the propellers do not break during impact, because of the uncontrollable rise in the net torque acting on the drone, it is bound to fall. The most crucial benefit of using a foldable/compliant body is that even if the robot crashes to the ground, it does not break and is ready for flight again.

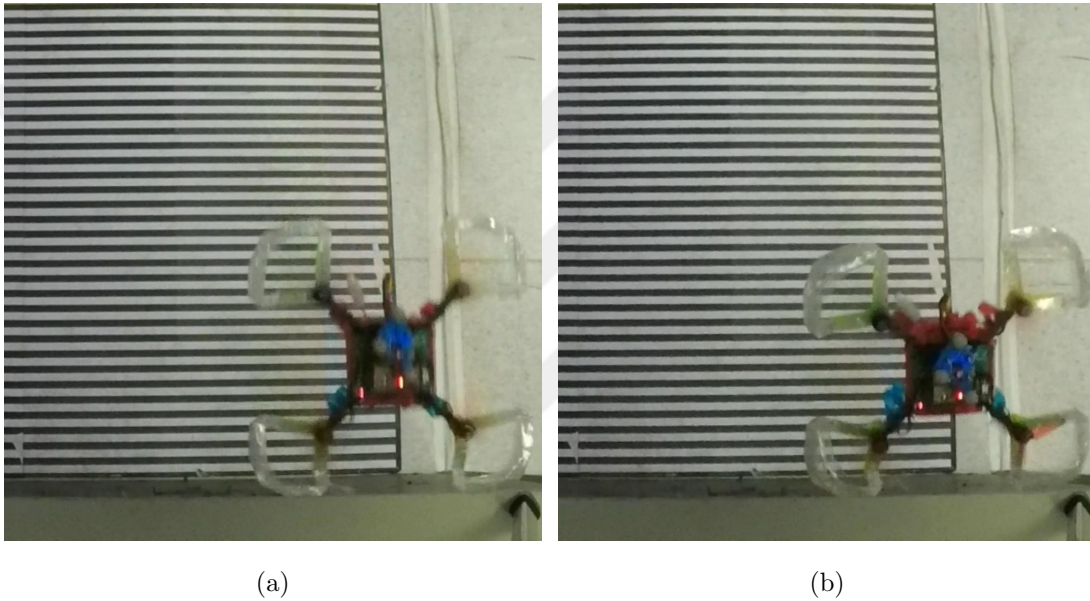


Figure 5.28: The robot with foldable bumper and foldable body crashing. (a) Bumpers flex during the crash, (b) Propellers momentarily stop

After testing both completely rigid and compliant drone bodies, we realized that combining both would bring the best of both worlds. Firstly, a rigid drone frame with a foldable bumper is tested. These tests were the least reliable ones as this combination brings the worst of both worlds; stopping the flexing of the drone arms and causing propellers to stop during impact. With this combination, the robot is able to endure impacts up to 1.7 m/s.

Finally, the drone body made out of compliant arms and rigid bumpers is tested, resulting in a highly durable robot that can bounce back and continue flying even after 3.0 m/s impacts occur. The ability to fly even after higher-speed collisions is mainly thanks to the rigid bumpers, as they ensure all the propellers keep turning during the impact so that the robot can control itself while the arms

flex. The drone body changes its shape dramatically during collision; however, the controllers onboard keep working perfectly since the shape change is for a brief time.

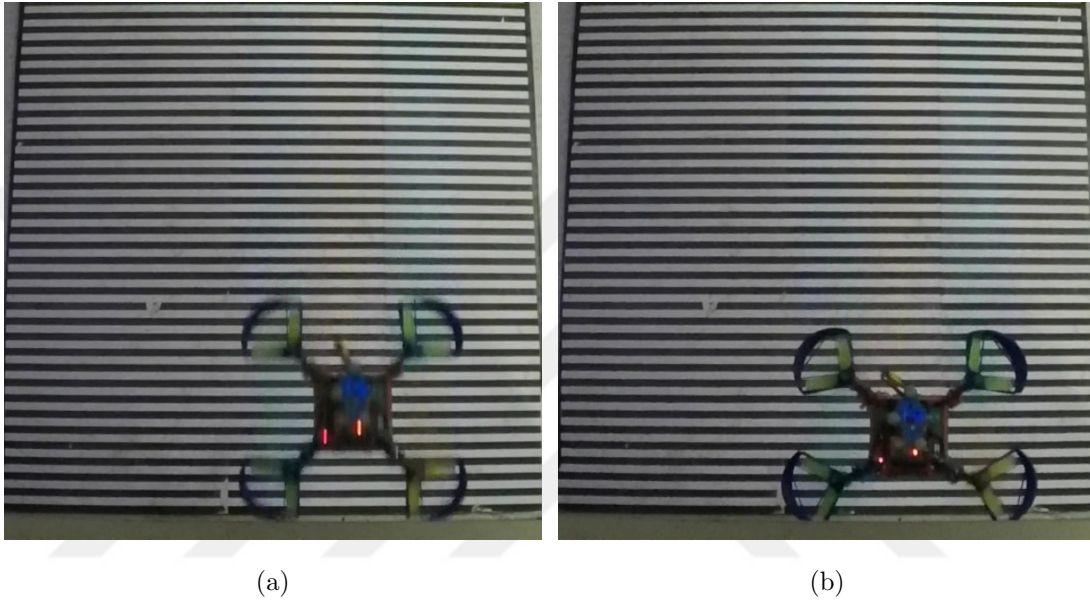


Figure 5.29: The robot with foldable bumper and foldable body crashing. (a) Bumpers flex during the crash, (b) Propellers momentarily stop

In summary, the experiments showed that simply introducing compliance to the whole frame does not contribute to the overall impact performance of the robot. However, it can increase the survivability of the drone even if it crashes. Instead of making the whole robot compliant, introducing rigidity to the different parts of the drone, such as the bumpers, is essential for both protecting the vulnerable points of the drone and still having the damping effect of the compliant nature of the foldable drone. Compared to the rigid design, combining the rigid bumpers and compliant arms increased the maximum impact speed the drone can endure by more than 50%.

Ultimately, experiments are designed so that the drones hit the wall with a predefined speed under a motion capture system. Every test flight was done at least five times for every impact speed for the reliability of the data, and the tests started from 0.1 m/s with 0.1 m/s increments up to the speed at which the

drone breaks or crashes. The resultant data is interesting as the worst design was the one utilizing foldable bumpers on the rigid body withstanding impacts up to 1.7 m/s. Close third- and second-best designs were the completely rigid body and completely foldable body, respectively. The former has a maximum impact speed of 1.8 m/s, while the latter has 1.9 m/s. Even though surviving from these impacts is impressive for a miniature drone, we managed to increase the maximum impact speed from which the miniature drone can survive to 3 m/s with the design utilizing all-around protection of the rigid bumpers to prevent propellers touching anywhere and the damping effects of the compliant body.

Chapter 6

Conclusion and Future Work

In this study, a novel flying robot that combines the advantages of miniature and compliant robots was designed and manufactured. In the robot design, PET sheets were used as the base material of the body with origami-inspired foldable robot techniques. The results show that the utilization of compliant materials in the robot's construction enhances its collision resistance, ensuring it remains intact during transportation and can be conveniently carried in a backpack. However, the project revealed that the problems regarding the robot being miniature and lightweight were underestimated at the initial stages. The miniature dimensions of the robot accelerate its dynamics and make it difficult to control the robot. While the robot can be controlled as the controller speeds can be increased as desired in the simulation environment, the limited control loop speed in reality, poses a challenge to the robot's flight capabilities. In addition, since the inertia of the robot is relatively low, the position controller of the robot has difficulty in windy environments. The robot is instantaneously dragged by the wind, which makes it difficult for the position controller to operate with high precision. Therefore, in order for such lightweight and miniaturized robots to be used in real-life scenarios such as an infrastructure imaging task, there is a need for improvements in the controller and robot electronics.

The existing literature offers limited insights into miniature and flexible flying

robots, emphasizing the significance of the work conducted in this study. For this reason, the work done here can lead to more critical publications and pioneer the production of miniature robots that are more capable, compliant, and well-designed. In addition, the collision capabilities of the robot were extensively investigated, with a particular focus on the influence of the robot’s flexibility on collision dynamics. Furthermore, flexible sensors were integrated into the robot, and the robot detected the collisions.

Future work will involve the utilization of soft sensors as structural elements to improve the collision dynamics of the robot. We want to optimize the stiffness of the soft sensors placed by changing the cross-sectional area, length of the sensors, and the number of coils. The sensor characterization experiments and the collision model of the SCoReR will be necessary to optimize the stiffness of the soft sensors by changing sensor geometry and sensor placement. Using the soft sensors as a structural element on the quadcopter will benefit the drone when a collision occurs, helping passively dampen the collision. The primary motivation for the future is to provide low-cost and feasible solutions to be one step closer to a miniature sensorized foldable collision-resilient quadcopter. In addition, the collision-resilient quadcopter will be used in a case study where it carries the ***Reconfigurable, Foldable, and Miniature Robot (ReMBot)*** if required. When the REMBOT faces an unscalable wall even with multiple modules, the collision resilient quadcopter can get into confined places and pick up REMBOT to carry it to a mission objective.

Bibliography

- [1] P. M. Kornatowski, S. Mintchev, and D. Floreano, “An origami-inspired cargo drone,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6855–6862, IEEE, 2017.
- [2] R. de Azambuja, H. Fouad, Y. Bouteiller, C. Sol, and G. Beltrame, “When being soft makes you tough: A collision-resilient quadcopter inspired by arthropods’ exoskeletons,” in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 7854–7860, IEEE, 2022.
- [3] J. Shu and P. Chirarattananon, “A quadrotor with an origami-inspired protective mechanism,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3820–3827, 2019.
- [4] P. Sareh, P. Chermprayong, M. Emmanuelli, H. Nadeem, and M. Kovac, “Rotorigami: A rotary origami protective system for robotic rotorcraft,” *Science Robotics*, vol. 3, no. 22, p. eaah5228, 2018.
- [5] L. Dilaveroğlu and O. Özcan, “Minicore: A miniature, foldable, collision resilient quadcopter,” in *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 176–181, IEEE, 2020.
- [6] S. Mintchev, S. de Rivaz, and D. Floreano, “Insect-inspired mechanical resilience for multicopters,” *IEEE Robotics and automation letters*, vol. 2, no. 3, pp. 1248–1255, 2017.
- [7] D. Yang, S. Mishra, D. M. Aukes, and W. Zhang, “Design, planning, and control of an origami-inspired foldable quadrotor,” in *2019 American Control Conference (ACC)*, pp. 2551–2556, IEEE, 2019.

- [8] A. Abazari, *Design, fabrication, and soft impact modeling and simulation of a collision-resilient foldable micro quadcopter*. PhD thesis, Bilkent University, 2022.
- [9] Y. Mulgaonkar, A. Makineni, L. Guerrero-Bonilla, and V. Kumar, “Robust aerial robot swarms without collision avoidance,” *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 596–603, 2017.
- [10] N. Khedekar, F. Mascarich, C. Papachristos, T. Dang, and K. Alexis, “Contact-based navigation path planning for aerial robots,” in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 4161–4167, IEEE, 2019.
- [11] A. Briod, P. Kornatowski, A. Klaptocz, A. Garnier, M. Pagnamenta, J.-C. Zufferey, and D. Floreano, “Contact-based navigation for an autonomous flying robot,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3987–3992, IEEE, 2013.
- [12] Z. Liu and K. Karydis, “Toward impact-resilient quadrotor design, collision characterization and recovery control to sustain flight after collisions,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 183–189, IEEE, 2021.
- [13] Y. Mulgaonkar, W. Liu, D. Thakur, K. Daniilidis, C. J. Taylor, and V. Kumar, “The tiercel: A novel autonomous micro aerial vehicle that can map the environment by flying into obstacles,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7448–7454, IEEE, 2020.
- [14] D. Özbek, T. B. Yılmaz, M. A. İ. Kalın, K. Şentürk, and O. Özcan, “Detecting scalable obstacles using soft sensors in the body of a compliant quadruped,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1745–1751, 2022.
- [15] W. Tabib, K. Goel, J. Yao, C. Boirum, and N. Michael, “Autonomous cave surveying with an aerial robot,” *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1016–1032, 2021.

- [16] F. Xiao, P. Zheng, J. di Tria, B. B. Kocer, and M. Kovac, “Optic flow-based reactive collision prevention for mavs using the fictitious obstacle hypothesis,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3144–3151, 2021.
- [17] H. Beck, J. Lesueur, G. Charland-Arcand, O. Akhrif, S. Gagné, F. Gagnon, and D. Couillard, “Autonomous takeoff and landing of a quadcopter,” in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 475–484, IEEE, 2016.
- [18] A. Bakir, D. Özbek, A. Abazari, and O. Özcan, “Scorer: Sensorized collision resilient aerial robot,” in *2023 IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 1–7, IEEE, 2023.
- [19] T. Lee, M. Leok, and N. H. McClamroch, “Geometric tracking control of a quadrotor uav on $se(3)$,” in *49th IEEE Conference on Decision and Control (CDC)*, pp. 5420–5425, 2010.
- [20] V. Praveen, S. Pillai, *et al.*, “Modeling and simulation of quadcopter using pid controller,” *International Journal of Control Theory and Applications*, vol. 9, no. 15, pp. 7151–7158, 2016.
- [21] K. Patel and J. Barve, “Modeling, simulation and control study for the quad-copter uav,” in *2014 9th International Conference on Industrial and Information Systems (ICIIS)*, pp. 1–6, IEEE, 2014.
- [22] P. Pounds, R. Mahony, and P. Corke, “Modelling and control of a large quadrotor robot,” *Control Engineering Practice*, vol. 18, no. 7, pp. 691–699, 2010.
- [23] N. Bao, X. Ran, Z. Wu, Y. Xue, and K. Wang, “Research on attitude controller of quadcopter based on cascade pid control algorithm,” in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1493–1497, IEEE, 2017.
- [24] S. González-Vázquez and J. Moreno-Valenzuela, “A new nonlinear pi/pid controller for quadrotor posture regulation,” in *2010 IEEE Electronics, Robotics and Automotive Mechanics Conference*, pp. 642–647, 2010.

- [25] O. Liang, “Looptime and flight controller.” <https://oscarliang.com/best-looptime-flight-controller/>, November 2016.
- [26] I. M. S. Provider, “Ist8310 3d magnetometer datasheet.” <https://intofpv.com/attachment.php?aid=8104>, cited January 2022. Version 1.2.
- [27] T. E. Connectivity, “Ms5611-01ba03 barometric pressure sensor, with stainless steel cap.” <https://www.amsys-sensor.com/downloads/data/MS5611-01BA03-AMSYS-datasheet.pdf>, June 2017. MS5611-01BA03.
- [28] I. SiRF Technology, “Nmea reference manual.” <https://www.sparkfun.com/datasheets/GPS/NMEA%20Reference%20Manual-Rev2.1-Dec07.pdf>, December 2007.
- [29] M. Ozkan, “Latitude,longitude,height // xyz converter.” <http://jsfiddle.net/mozkan/nz1otcaq/>, Cited January 2022.
- [30] M. Kleder, “Covert lat, lon, alt to ecef cartesian.” <https://www.mathworks.com/matlabcentral/fileexchange/7942-covert-lat-lon-alt-to-ecef-cartesian>, July 2005.
- [31] E. Systems, “Esp-now user guide.” https://www.espressif.com/sites/default/files/documentation/esp-now_user_guide_en.pdf, July 2016.

Appendix A

Data

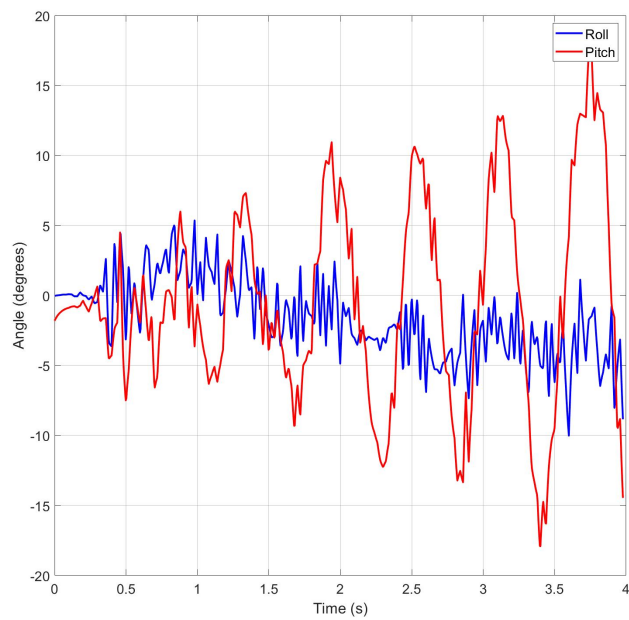


Figure A.1: Simple PID controller with $K_p=0.001$, $K_i=0$, $K_d=0$

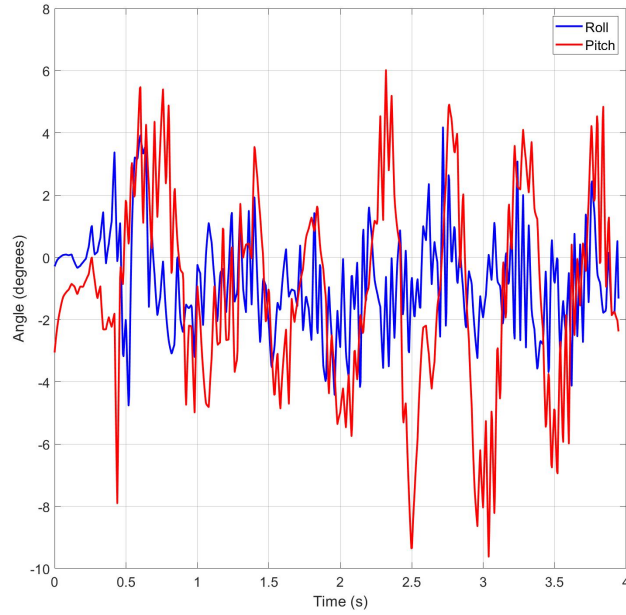


Figure A.2: Simple PID controller with $K_p=0.01$, $K_i=0$, $K_d=0$

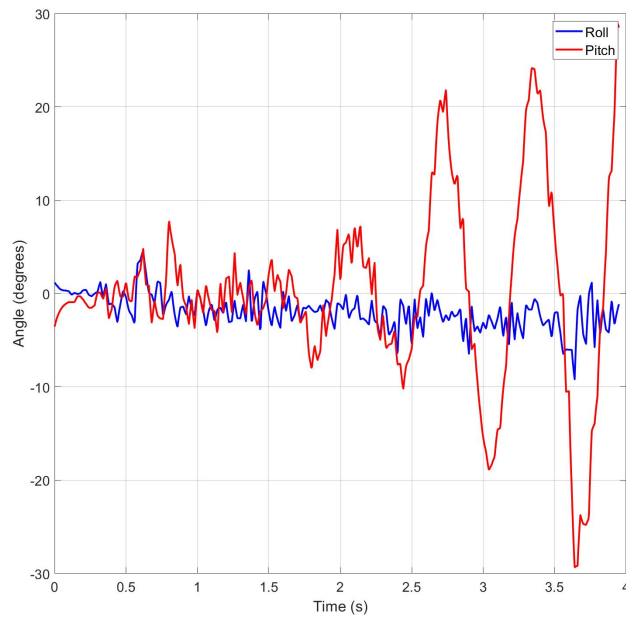


Figure A.3: Simple PID controller with $K_p=0.1$, $K_i=0$, $K_d=0$

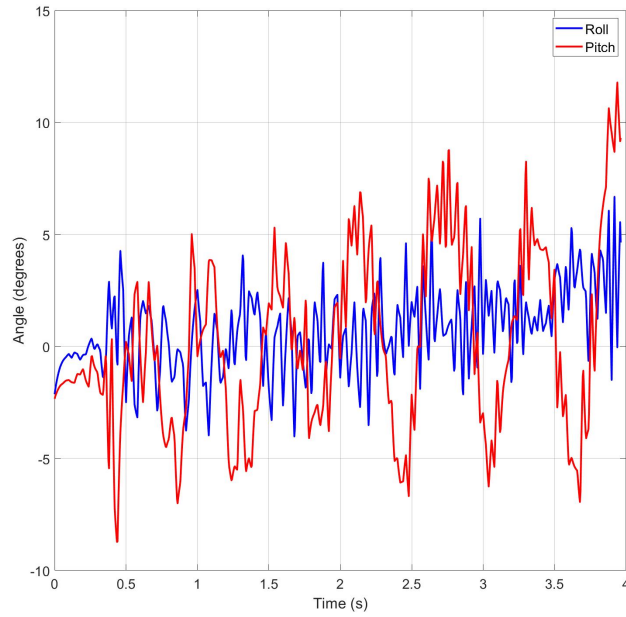


Figure A.4: Simple PID controller with $K_p=1$, $K_i=0$, $K_d=0.1$

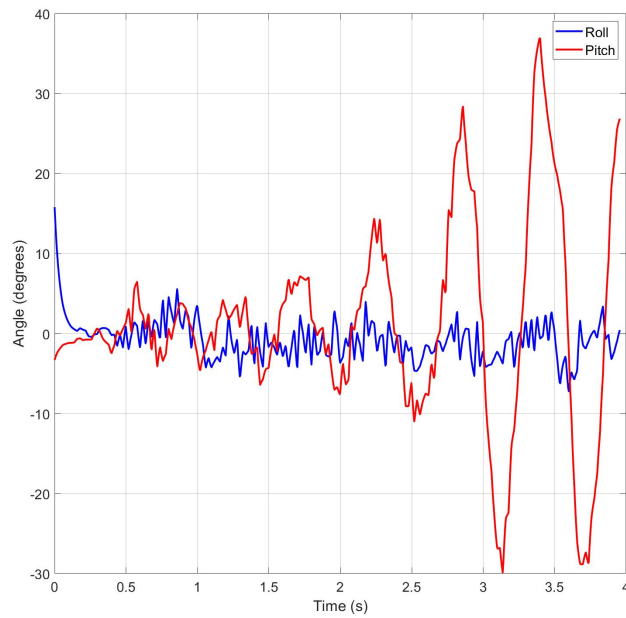


Figure A.5: Simple PID controller with $K_p=1$, $K_i=0$, $K_d=0.5$

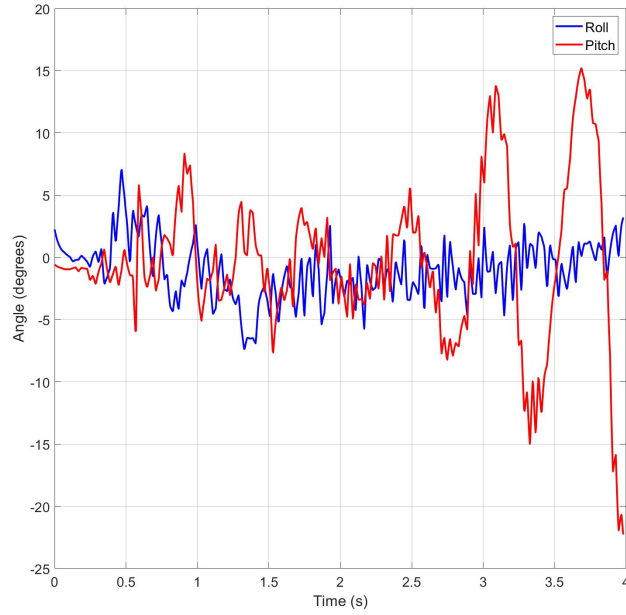


Figure A.6: Simple PID controller with $K_p=1$, $K_i=0$, $K_d=0$

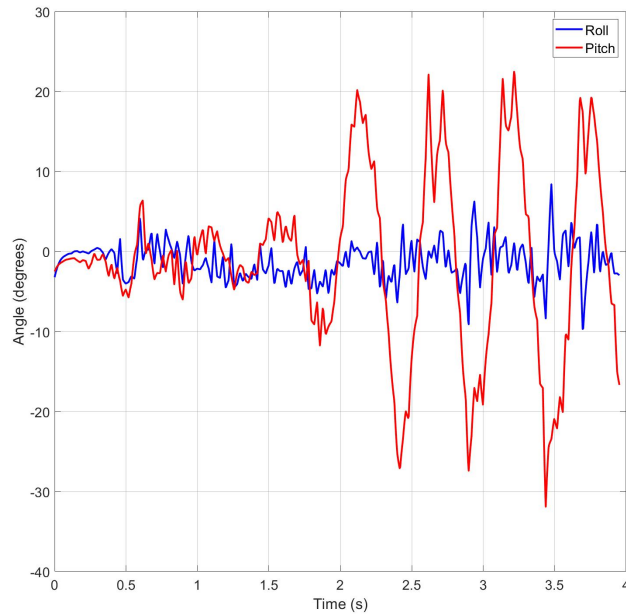


Figure A.7: Simple PID controller with $K_p=1$, $K_i=0$, $K_d=1$

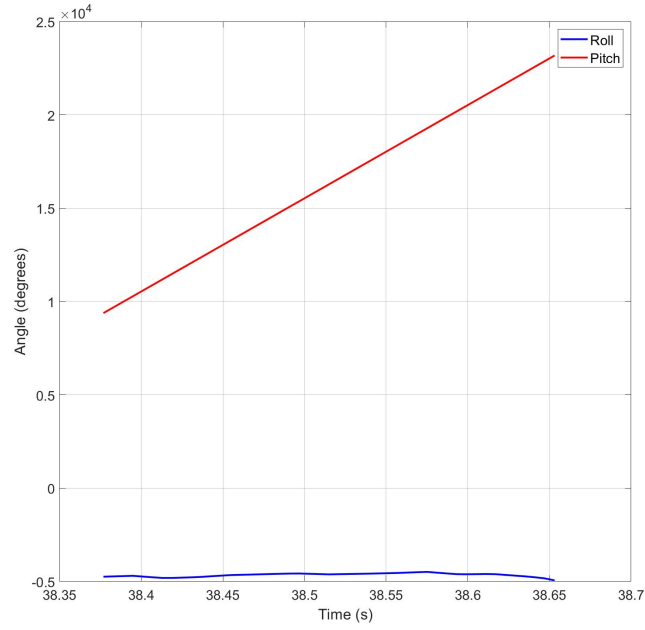


Figure A.8: Cascaded PID controller with $K_p^0=1$, $K_i^0=20$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)

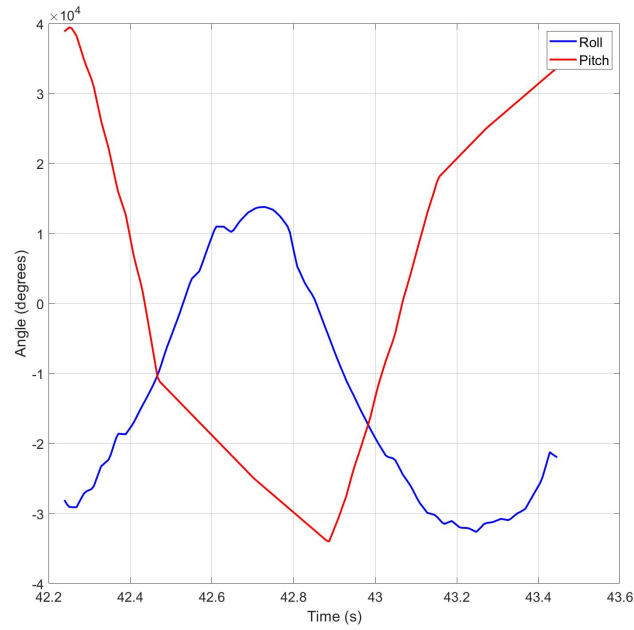


Figure A.9: Cascaded PID controller with $K_p^0=10$, $K_i^0=20$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)

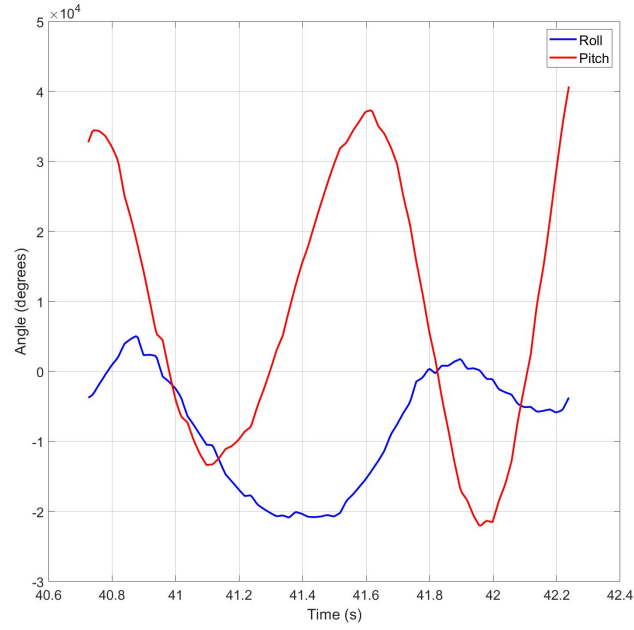


Figure A.10: Cascaded PID controller with $K_p^0=15.2$, $K_i^0=10$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)

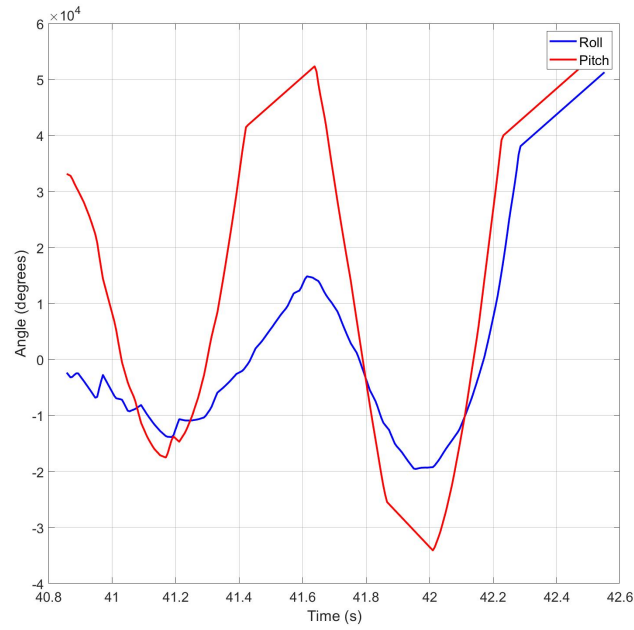


Figure A.11: Cascaded PID controller with $K_p^0=15.2$, $K_i^0=20$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)

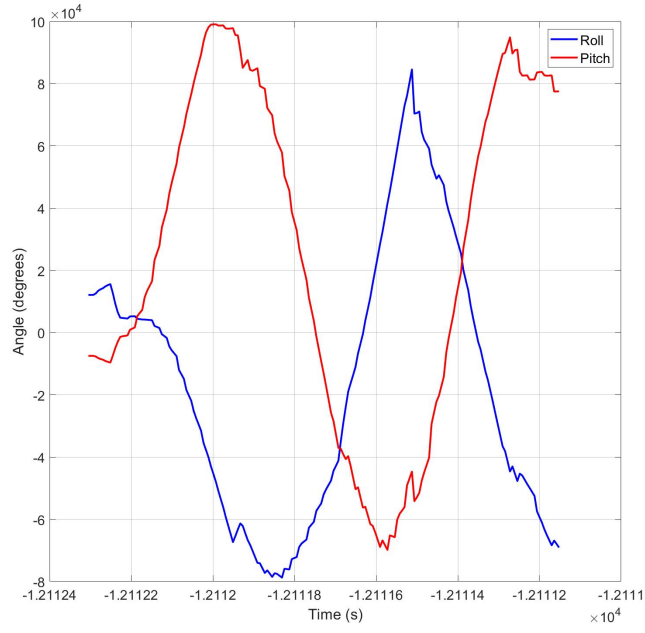


Figure A.12: Cascaded PID controller with $K_p^0=30$, $K_i^0=0.1$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)

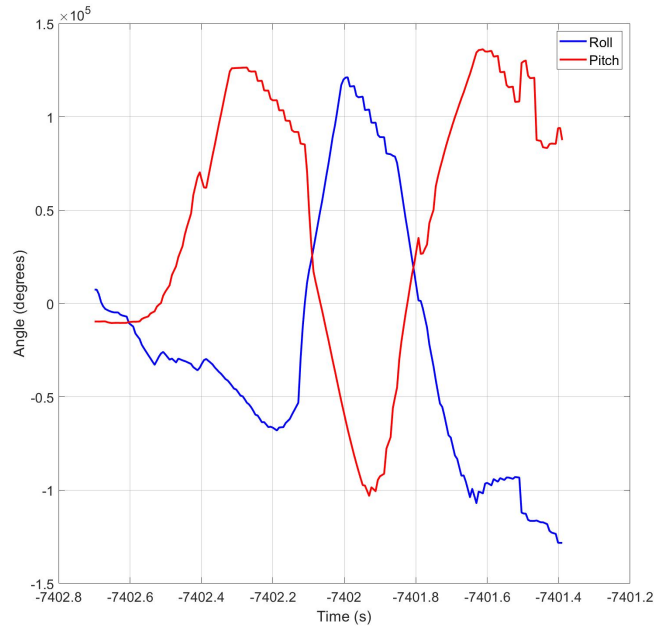


Figure A.13: Cascaded PID controller with $K_p^0=40$, $K_i^0=1$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)

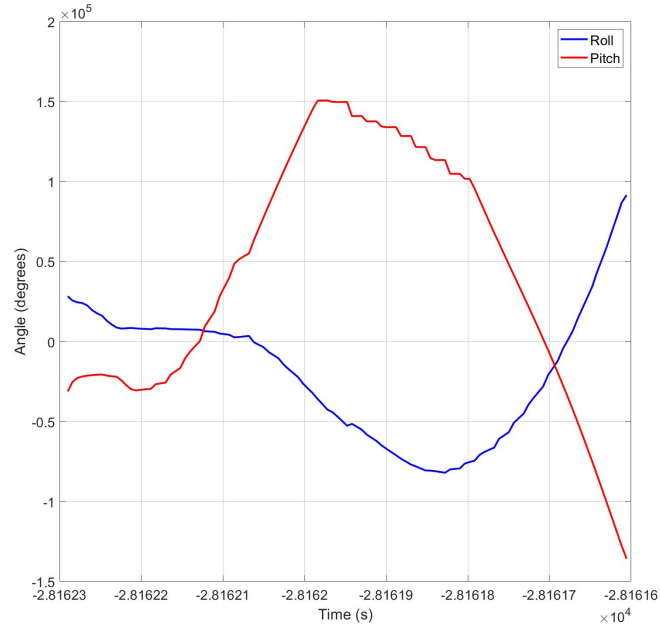


Figure A.14: Cascaded PID controller with $K_p^0=50$, $K_i^0=1$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)

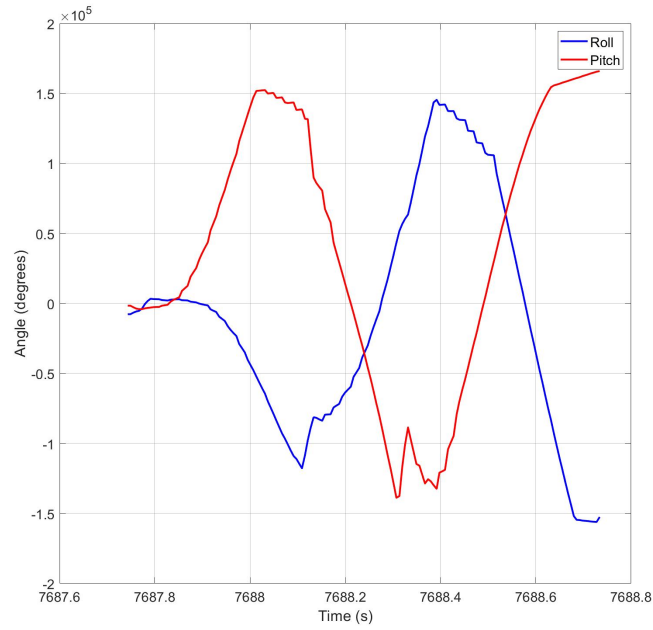


Figure A.15: Cascaded PID controller with $K_p^0=50$, $K_i^0=10$, $K_p=0.01$, $K_d=0.1$. (Angle is multiplied by 100)

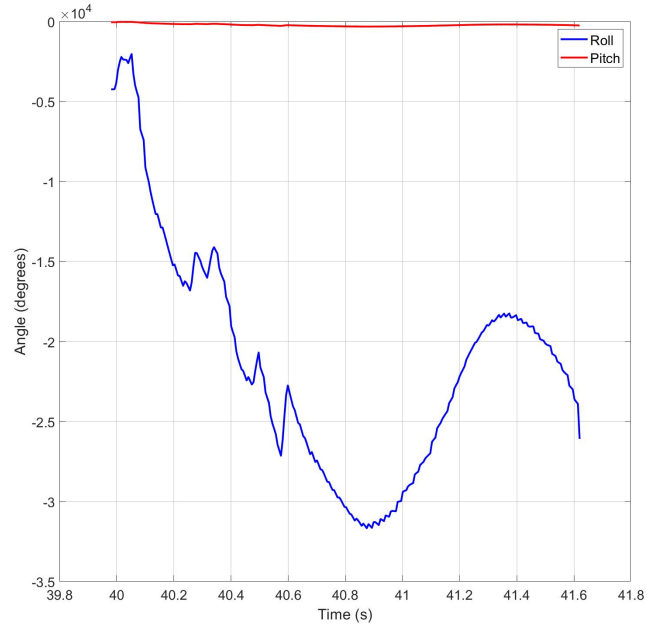


Figure A.16: Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.01$, $K_d=0.05$. (Angle is multiplied by 100)

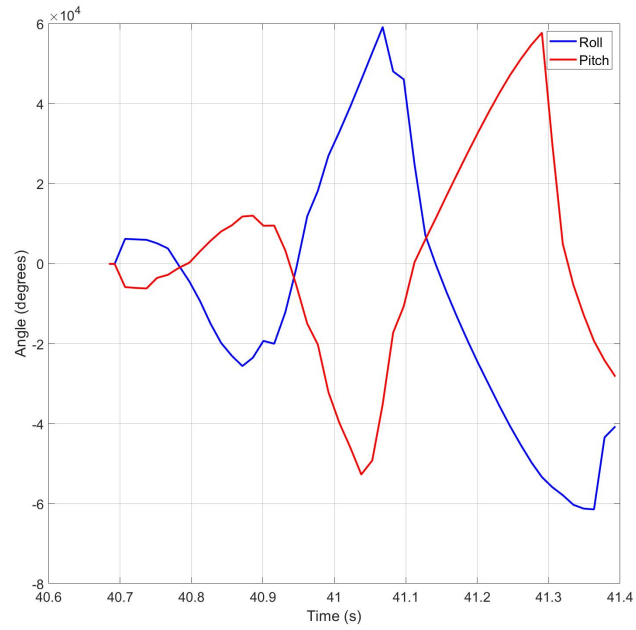


Figure A.17: Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.1$, $K_d=0.05$. (Angle is multiplied by 100)

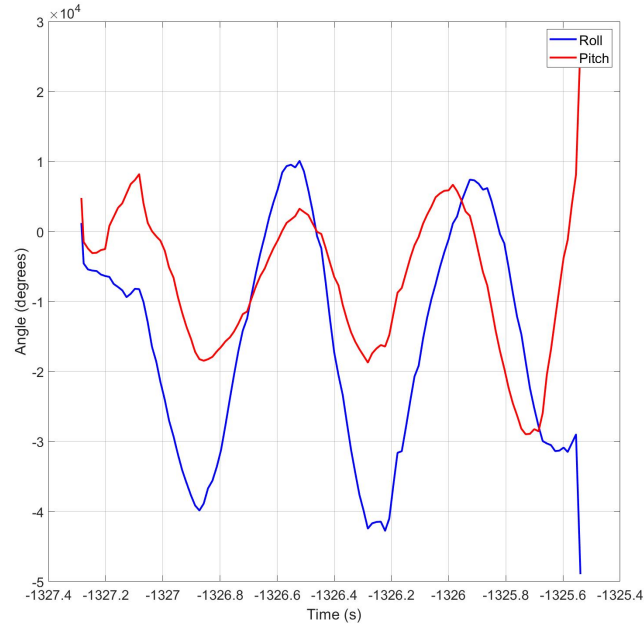


Figure A.18: Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.02$, $K_d=0.05$. (Angle is multiplied by 100)

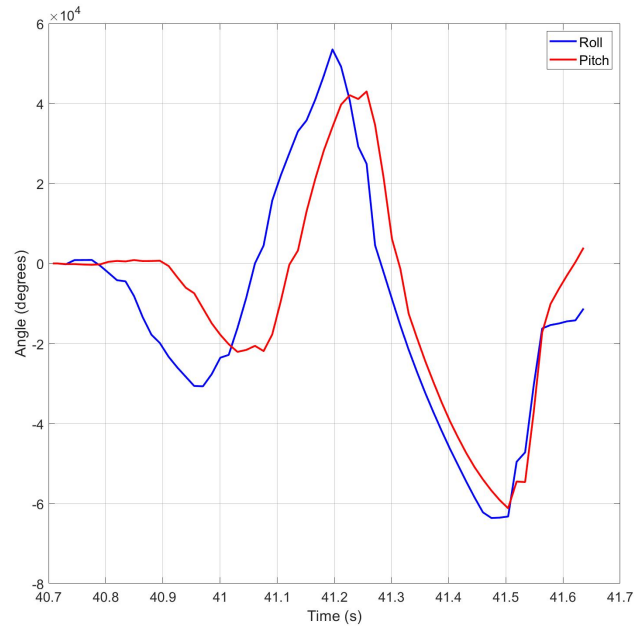


Figure A.19: Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.05$, $K_d=0.05$. (Angle is multiplied by 100)

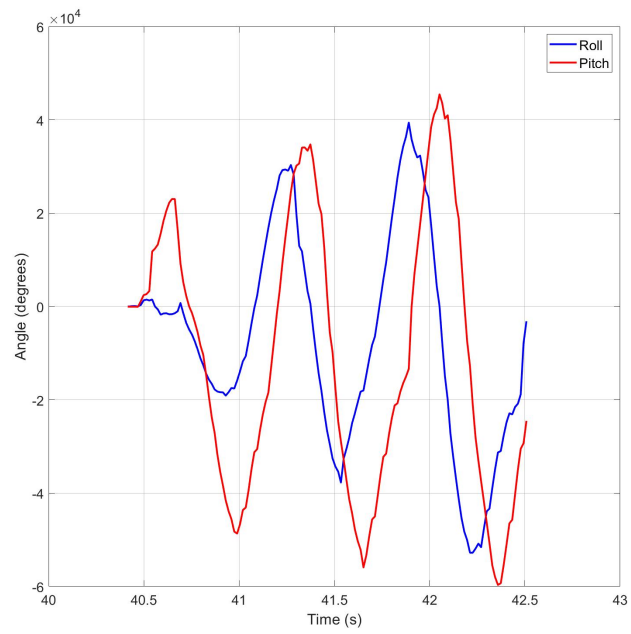


Figure A.20: Cascaded PID controller with $K_p^0=15.2$, $K_i^0=0.1$, $K_p=0.015$, $K_d=0.05$. (Angle is multiplied by 100)