POLITECNICO DI TORINO

Master Degree Course in Mechatronic Engineering

Master Degree Thesis

An Off-board Model Predictive Control With Obstacle Avoidance For Unmanned Aerial Vehicles

Supervisors
Prof. Alessandro Rizzo
Dr. Stefano Primatesta

Candidate
Kamil Umur Güzel

July, 2019

To my beloved family...

Summary

The Unmanned Aerial Vehicle technology, which was supported for the development of military technology in the early periods, has become one of the most challenging engineering research topics with the recent development of microcomputer, sensor and battery technology. The UAV field of application has been expanding relative to the increase in the capability and the feasibility of the UAV technology itself.

The aim of this thesis is to design an off-board linear Model Predictive Control with obstacle avoidance, improving vehicle capability. In fact, the control problem is split into two parts: an attitude controller and a motion controller. The attitude controller is provided by a low-level controller on-board (e.g. the on-board autopilot). Hence, off-board, the motion controller provides an optimal trajectory. In this thesis, the off-board controller is performed with a linear Model Predictive Control with obstacle avoidance capabilities. According to the goal of the project, the development of the UAV is followed from the beginning to the current stage in order to understand the role in the challenging engineering problems. The UAV concept, components and the control theory are comprehensively covered and discussed during this thesis report. The theory of the linear model predictive control, which is based on the predictions of the future steps by considering the mathematical model of the plant, is examined and combined with the Object Oriented Programming skills based on the C++ language in order to develop and improve the control of the vehicle which is selected as quadcopter for this research. In particular, the proposed control strategy is implemented to be used in the Robotic Operating System(ROS) framework. The mathematical model of the vehicle is evaluated to have proper representation of the system in order to increase the accuracy and the control. On the other hand, the obstacle avoidance capability is derived thanks to the adopted Mixed Integer Linear Programming(MILP) technique which is employed in the our ROS-based implementation. For that implementation, the convex optimization procedure is operated with the selected solver which is developed in Python language thanks to the compatibility benefit of the ROS framework. Thus, the implemented approach is tested with the different target points which are selected in order to test capability limits and the results are discussed in the sense of the feasibility and performance in the last chapter.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The first record of the unmanned aerial vehicles (UAV) in the history was 22 August 1849, The Austrian incendiary balloon attack on the Venice, Italy. The balloons were designed to perform air bombardment with the guidance of smaller pilot balloons. However,by the virtue of the fluctuating winds,the operation targets were missed. Then a quite few balloons drift back over the Austrians [1]. That military operation was failed but it was just the beginning for the Unmanned Aerial Vehicle concept to be noticed in further.

In the early 20th century, the military funding of science has had a powerful impact on the progress and practise of scientific researches. In 1916, when the world war I was about the end , the first pilotless radio controlled unmanned aerial vehicle which is "Ruston Proctor Aerial Target" made its first flight on the sky. It was designed to used in ramming strikes against hydrogen filled Zeppelins[2].

Even at the time that the aerial vehicle technology has been starting to develop, there was no doubt that the aerial superiority of an air force of the army was significant military advantage. The first gyroscopes controlled unmanned aerial vehicle project which is "Hewitt-Sperry Automatic Airplane" completed in Britain in September 1917 [3]. It was designed to use as aerial torpedoes. In October 1917, the "Hewitt-Sperry Automatic Airplane" project is purchased and upgraded by The US Army then it called as "Kettering Bug" which took off in October 1918 [4].

Figure 1.1: Kettering Bug.

In the end of the WWI, The US Army noticed the promising concept of UAV. Hence they decided to convert three of the standart E-1 fighter aircraft to UAV [5]. In 1927-1929 the Royal Navy which is United Kingdom naval warfare force started to test of their pilotless auto guided anti-ship aircraft "Larynx"[6]. The development of the radio technology led up new capabilities. When it comes to 1931 The British aircraft company which is "De Havilland Aircraft" attained fascinating commercial success by the light weight training aircraft model named "Tiger Moth". It was the first commonly used aircraft for the training of the military pilots[7].

In 1935 the new radio technology implemented on "Tiger Moth" aircraft then they introduced new radio controlled anti-aircraft gunner "DH.82B Queen Bee" . The radio signals used to control servomechanisms on the aircraft. But also the name father of the word of the Drone that we use for UAV in general use. Since the drone is the name for male bee which makes his last flight to find queen bee[8].

Figure 1.2: "DH.82B Queen Bee".

In 1940 , the first mass drone production made in United States by Radioplane Company. Approximately 15.000 unit of radio controlled target drone planned to use in US army for the anti-aircraft gunnery during the WWII. After the WWII drones were modernized with the jet engines and increased their capabilities. In the end of the 1950, The US army started to support drone progress in order to reduce pilot loss in military operations. At the same time development of the camera

technology open the new gate for the UAV. Advance in military surveillance and reconnaissance became very significant capability during the Cold War. For that reason drones were evaluated by equipping camera system to achieve intelligence[9]. Surveillance missions provided by pilots who were trained on control of drone to fly on different targets to scan and monitor locations.

The first modern military UAV which is "RQ-1 Predator" built by the support of the United States Department of Defence in 1995. Development of the sensor technology and sophisticated precision control on the RQ-1 predator could provided reliable information about ground situation more accurate than in any previous aerial vehicles.

Figure 1.3: "RQ-1 Predator".

Although the driving force behind the UAV history of progress was military purpose, the interest of use increasing through different areas such as commercial, scientific, recreational, agricultural, policing, peacekeeping, product deliveries, aerial photography, smuggling and drone racing[10]. The Federal Aviation Administration of US started to give the initial commercial drone permit to civilians in 2006, also some other agencies introduced new drone systems for the different tasks rather than use in military. Because the drone capability would offers new solutions to problems in disaster emergency situations,rescue, border controls, commercial delivering systems, firefighting and agricultural spraying.

1.2 Problem and motivation

Current technological advance in capability of the sensor,performance of the processor and the efficiency of the battery leads up the progress of the unmanned aerial vehicles in the recent years. The variety of use of the UAV is enlarging day by day thanks to the growing interest not only in military use but also in the research of robotic science and the commercial applications. The sufficient agility in 3 dimension space and the endurance during the flight under the reasonable payload open the gate for new operational areas. In particular, the use of UAV in enclosed industrial inspection tasks can be good solution in order to avoid from dangerous situations. [11]. Moreover, the use of the small scale of UAV offers great performance in the sense of the practical convenience for the experimentation and feasible

cost for the implementation. Quadcopter is one of the type of the small scale unmanned rotorcraft that has 4 propeller located at the vertices of a square frame and two pairs counter-rotating rotors [12]. These advantages may speed up the research of autonomous capability in aerial robotics. But the small scale model has richer dynamics rather than typical UAV. In addition to that, the small scale of UAV has increased sensitivity to control inputs and disturbance [13]. The appropriate sensors selection and the robust, reliable, effective control design become very fundamental issue of UAV, since it has the highly unstable and nonlinear behaviour[14]. Some of the researchers developed controller by adopted different control approaches. In the [15], the PID control and linear quadratic control approaches are compared and developed for the real quadrotor application. It is found that general stabilization is highly affected by propellers speed control. Especially, classical PID approach has better performance than linear quadratic control approach when the quadrotor tries to stabilize at the hovering condition. The performance difference of the two approaches is obviously related with the model deficiency. However both of the control approaches are not sufficient if the strong perturbations like strong wind exist on the device. Moreover, some other suitable control approaches are implemented for the control of quadrotor. Sliding-mode control is one of the suitable commonly use approach for the nonlinear systems. But when it comes to real application, high frequency of the switching mode control signals would trigger the unmodeled dynamics of the real system, consequently, undesired oscillations which are named as chattering can appear. To avoid this, it is important to consider also sensors and actuators dynamics which are relatively faster than system dynamics for the system modelling [16]. In the real application of sliding mode control of the quadrotor shows that even the chattering appears in the control signal, the stabilization at the hovering condition is maintained with this approach where the roll, pitch and yaw angles converged into zero[17]. Contrarily, challenging initial condition of the yaw angle drives the system to have big negative overshoot in the pitch angle. Another possible suitable approach is backstepping control which is designed for nonlinear dynamics systems. Even though, the hard initial conditions initialized, it is capable to converge the roll, pitch, yaw angles into zero. But in that case, some angle drift happens on the control signal of the yaw angle [17]. Linear quadratic regulator approach has satisfactory results in the optimization of the closed loop control whereas the assumptions mismatches with the real conditions. Because of the fact that the quadcopter dynamical model has strong non-linearities. Some different control approaches can also be adopted, but the most convenient control can only be obtained through the best performance in the real application scenarios. For that reason, disturbance effects and physical constraints should take into account for the robust and reliable control. These disturbance and constraints limitations can be directly calculated in each iteration step ahead in the model predictive control optimization procedure[18]. The model predictive control applied as hierarchical divided system where the linear model predictive control used for the stabilization tasks and and hybrid model predictive control employed to prevent drone from any collision by the on-board generated trajectory[19]. On the other hand, the real application performance under the strong disturbance conditions have not verified yet for hierarchical model predictive control approach [18].

Based on the given evaluated control strategies, the obstacle avoidance motion control capability will be implemented on The Linear Model Predictive Control approach for the quadcopter and the adopted approach performance will be discussed and tested through the varying scenarios in this project.

1.3 Outline of this thesis

In this thesis, we started with the history of Unmanned Aerial Vehicles and then the problem of controlling unmanned aerial vehicles and the our motivation are covered in the **Chapter 1**. One of main concept which is Robotic Operating System will be discussed and the further details will be given in the **Chapter 2**. Moreover, we will discuss the mathematical notations which will be involved during the report and project in the **Chapter 2**. Some of the important concepts about the mathematical modelling will be covered too. After that, we will give brief introduction and more detailed knowledge about the model based predictive control theory. One of the most important topic which is objective function will be discussed in the same chapter. When we comes in to the **Chapter 3**, we will describe the adopted control approach in a comprehensive way by explaining from the quadcopter dynamic model to the subsystems. In the same chapter, we will also discuss and introduce the robotic operating system integration of the project by us and also our software implementation will be shown in the same chapter. In the **Chapter 4**, we will present the result that we will have about the performance and capability of the developed solution. We will show the test scenarios and the outputs by considering the computation time performance. Also some comments about the result,simulation and the error analysis will be placed in the same chapter as well. In the last **Chapter 5**, we will conclude the project, explain some limitations and give our opinions about the future works.

Chapter 2

Background

In this chapter, we will explain the basics of the Robotic Operating System(ROS) and then the basic notations which are used in the modelling will be covered. The derivation of the dynamic mathematical model which will be used by the control algorithm of the drone will be presented and the theory of the Model Predictive Control approach will be introduced.

2.1 Robot Operating System

It has been experienced that the software programming of the robotics applications occupied too much time during the development of the robotic technology in the past. Since the researchers and developers had to spend abundant amounts of time designing the embedded software within a robot and also the hardware itself. So that it could be imagined as the programming start with a driver level of the components to more higher level control and even artificial intelligence algorithms. In general, it is not easy to write code that can be implemented into various type of the platforms or robots, and the developers may become from different field of engineering disciplines as such as mechanical engineering, electronics and computer science. Additionally they may prefer to use different type of the coding languages which creates a code integration and interpretation problem in order to build robotic control and operating system for the complex applications.

For these reasons, robot operating system (ROS) is invented as a meta-operating system which can be positioned between operating system and a software which provides services to applications outside of capability of the operating system. It makes enable clear and brief communication links between robotic applications in order to minimize integration efforts and avoids from the repeatedly rewriting code. It creates a convenient way to use your developed code with different platforms, through different programming languages and different levels of the software by simplifying a work load to develop an experimental set and making it possible to perform the experiments in efficient way. It includes the common operating system capabilities such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes and package management. It also offers a convenient features thanks to the tools and wide range

of the libraries for collecting, developing and compiling code across not only in the individual computer but also with the multiple computers. The advantages of the ROS could be easily noticed in the stage of writing drivers to manage hardware, controlling the process and organizing the memory, controlling the availability of the data. The ideas behind the Robot Operating System can be listed as follows :

- The architecture is developed in a peer-to-peer topology, which makes enable a number of processes can be running in a single host computer or in multiple hosts computers and communicate to each other via standard UDP protocols. A peer-to-peer architecture coupled to a buffering system and a master process, enables each component to communicate directly with any other, synchronously or asynchronously as required. The coordination of the communication tasks is provided by a master process that can run in any computer in the network according to the your own preferences.
- It allows to write a node program in various type of programming languages. Since the ROS is a language-neutral. Also it has message describer which is named as Interface Definition Language (IDL), that characterise each field of the message for the code generators and compilers of each language to generate an implementation native to the related language,in order to support a new language, either C++ classes are re-wrapped or classes are written enabling messages to be generated.
- It is quite robust and flexible system because of the adopted decentralized run time environment system. This strategy can offer a convenient solution to a complex executing problem where one executable does not affect the others since each command is in fact an executable. Thanks to the adopted microkernel design in the ROS, it uses a large number of small tools to build and run the various ROS components.
- The drivers and other algorithms are placed in standalone executables. The reason behind this partition can be explained through the challenging in the development of algorithms that are entangled to a greater or lesser degree with the robotic operating system and are therefore hard to reuse subsequently. Thanks to the adopted approach, it guarantees maximum reusability and reduce the size but without wrapping the written code by developer.

2.1.1 ROS File System

In the ROS file system, it organizes resources files into a hierarchical structure on disc. The general structure can be seen in the Figure 1.4.

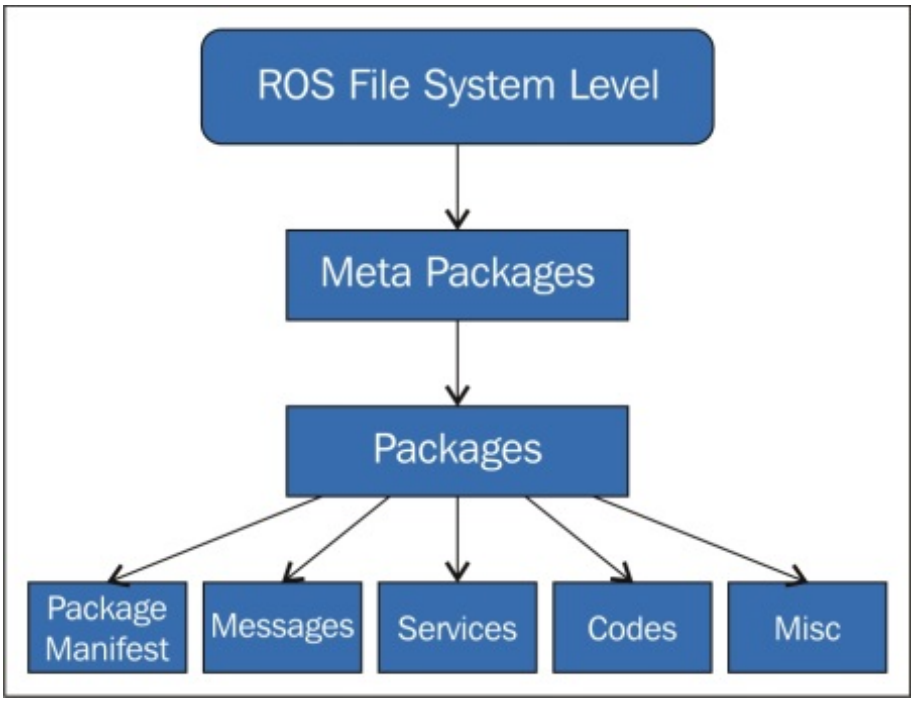

Figure 2.1: "ROS File Structure".

Some of the important concepts of the file system can be introduced as follows :

- **Packages:** Packages are the major unit for organizing software in ROS. Since, software in ROS is organized in packages. A package can hold ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that coherently constitutes a useful module. The objective of the package use is to provide this useful functionality in an easy-to-consume manner so that software can be easily reused again and again. In general, ROS packages are designed through a "Goldilocks" principle which means that the functionality to be convenient, but not too much that the package is over size and difficult to use from other software.

- **Metapackages:** Metapackages are designed particularly as Packages in ROS. They can be considered as collection of packages which create high level library. They do not required to install files and they do not contain any tests, code, files, or other items usually found in packages.A metapackage is used in a similar way as virtual packages are used in the debian packaging world. A metapackage simply references one or more related packages which are loosely grouped together.

- **Messages:** ROS uses a simplification on the messages description language for describing the data values that ROS nodes publish. So that the description creates convenient way for ROS tools to automatically generate source code for the message type which can be used by the various target languages. Message descriptions are located in ".msg" files in the "msg/" subdirectory of a ROS package. A ".msg file" can be divided into two section which are fields and

constants. Fields are the data that is sent inside of the message. Constants describe useful values that can be used to interpret those fields.

- **Package Manifest:** The package manifest is a type of XML file which is called as package.xml that must be involved with any catkin-compliant package's root folder. This file presents properties about the package, for instance, the package name, version numbers, authors, maintainers, and dependencies on other catkin packages. It is important to address that system package dependencies are declared in "package.xml". If they are missing or incorrect, it may be possible to build from source and run tests on your the machine, however, the package will not work correctly when released to the ROS community.

- **Services:** ROS performs a simplified service description language "srv" in order to describe the ROS service types. Moreover, it builds directly upon the ROS ".msg " format to enable request or response communication between the created nodes. Service descriptions are located in ".srv" files in the "srv/ subdirectory" of a package.

2.1.2 ROS Main Concepts

The fundamental goal of a robot operating system is to operate a considerable number of executable at the same time and also it should allow the data exchange synchronously or asynchronously in a communication manner. For instance, a robotics operating system needs to query sensors which are mounted on the robot. The sensor could be camera, accelerometer, temperature sensor, pressure sensor, gyroscope, distance sensor and any other sensors for the desired frequency. The operating system should should be able to capture sensor data, process it, pass it to processing algorithms which can be artificial vision, image process, simultaneous localization and mapping and more. Furthermore, it should control the motors in return. All the explained procedure is carried out continuously and in parallel. Also, the robotic operating system should operates contention to ensure efficient access to robot resources.

The given process and methods are collected and labeled inside the ROS Computation Graph. Some of the important concepts are described as in the below:

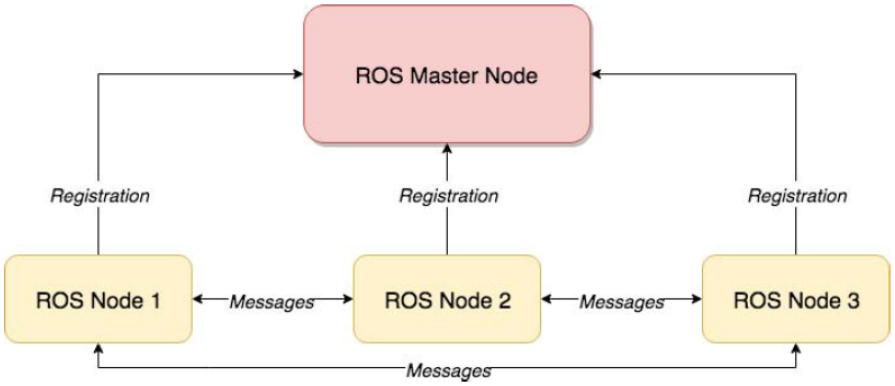

Figure 2.2: The relation of The Master Node with the other Node[1].

- **ROS Master Node:** The Master Node is a special node that is used for the declaration and registration service for the rest of the nodes. It pursues publishers and subscribers to topics. It is clear to say that without using master node, it is not possible to enable individual ROS nodes to locate one another and also it is not possible to have peer to peer communication in order to exchange the data between the nodes. The ROS master node also provides Parameter Server which can be seen as central database for the node data storing actions. The relation of the Master Node with the other nodes can be found in the Figure 1.5.

- **Topic:** A topic is designed for the data transfer actions where the messages are transmitted through a topic. A node is a responsible to publish a certain type of a message into the related topic. The relation between the node and the topic can be visualized as given as in the Figure. Furthermore, the great convenience of the topic that is able to identify the content of the message. When the topic is typed which means that the type of data published by the node which is a message is always structured in the same way since the nodes are regulated to send and receive messages on topics. It is possible to have multiple nodes are able to publish data to a topic and multiple nodes can read data on that topic. A topic can also be interpreted as an asynchronous message bus. This concept of an asynchronous, many-to-many bus is essential in a distributed system situation. Mostly, publishers and subscribers are not aware of each others existence. The idea is to manage the data transition in a efficient way to decouple the production of information from its consumption. In a nutshell, it is possible to consider the topic as a strongly typed message bus where each bus has a name, and anyone can connect to the bus to send or receive messages unless they are in the right type.

- **Message:** Nodes communicate with each other thanks to the passing messages. The context of the message can be the integer type, floating point type, Boolean type arrays or the combination which can be arbitrarily nested structures of them. For instance, in the Figure 1.6 , the basic nodes communication

[1]Image courtesy from Introduction to ROS, Clearpath Robotics,2015.

can be seen via topic feature and also the various type of the variables can be noticed inside the message context. The node 1 employed to publish the proper type of message into a topic and the node 2 which is designed as subscriber receive the message from topic itself. It is also possible to demonstrate this data transfer structure with a intuitive example. Robot servo motor can be introduced as a node, therefore, the context of the receiving messages can be floating type for the motor position and speed or Boolean type to check logical predefined constraints as the designer preferences.

- **Services:** Services are designed to maintain synchronous communication between two nodes. As it is explained in the topic section which is an asynchronous communication method used for many-to-many communication so that one way direction is not adequate for the request and reply interactions. However, these interactions are mostly expected in the distributed system. Services are used to maintain and manage request and reply interactions which can be seen in the Figure 1.7. Services are defined a pair of message structures in such a way that one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply.

- **Bags:** Bags are a format for storing and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms. For instance, to collect data measured by sensors and subsequently play it back as many times as desired to simulate real data. It is also a very useful system for debugging a system after the event.

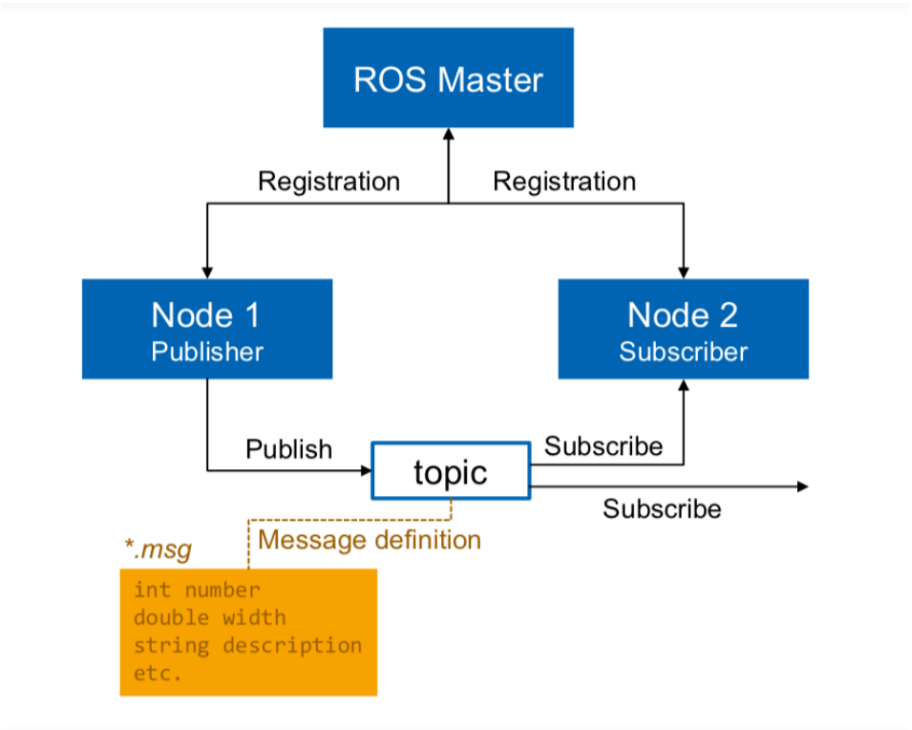

Figure 2.3: The relation of The Master Node with the other node and the example message[1].

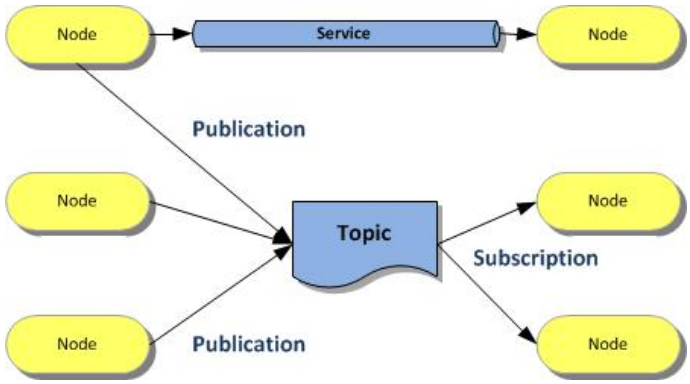

Figure 2.4: The relation of The Node via Service[1].

[1]Image courtesy from `https://wiki.ros.org/Messages`

[1]Image courtesy from `http://www.rignitc.com/introduction-to-ros/`

2.2 Notations

It is well known that it is required to use reference frame in order to describe the motion of the body in the space. For that reason we implement world fixed inertial frame in a 3-D space as a reference point to describe the subsequent motions of the body, as shown in the Figure 2.5. In addition, the origin of the body frame is fixed on the center of the gravity of the quadcopter.

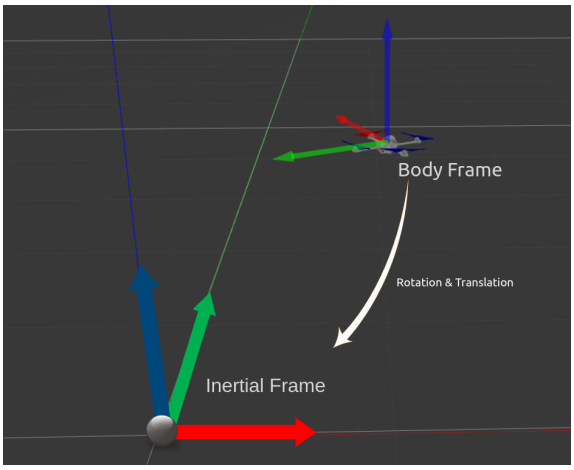

Figure 2.5: Inertial-Body Frames.

The velocity and the position of quad-copter's center of the gravity which is body frame origin can be describe in the inertial frame by using translation in the 3-D space. Additionally, the orientation of the quadcopter can be define in the inertial frame as well [18].

Tait-Bryan Euler Angle approached is adopted to define orientation of the quadcopter. These angles are also commonly known as roll, pitch and yaw angles [14]. The selected direction of the rotation axis are compatible with the roll, pitch and yaw conventions. Therefore, the counterclockwise rotation around the x axis is considered as positive angle and it is named as roll angle. Similarly, the counterclockwise rotation around the y axis can be considered as positive pitch angle. Lastly, the positive yaw angle can be determine by the counterclockwise rotation around the z axis. All the explained angles are represented in the Figure 2.6.

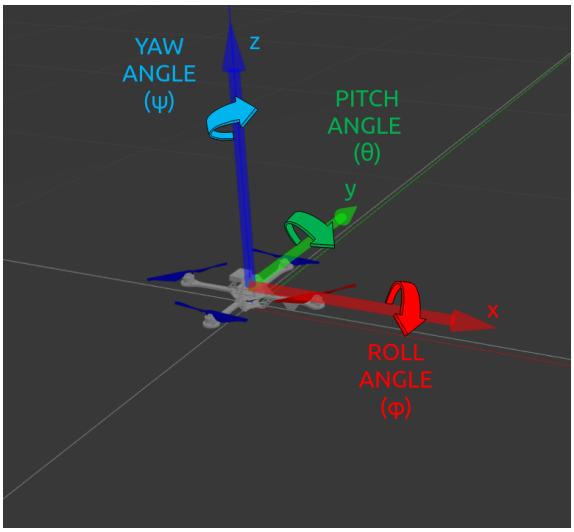

Figure 2.6: Roll,Pitch and Yaw Angles.

The quadcopter motions are created by the changing rotational speed rates of the rotor propellers. Naturally, quadcopter is built by 4 rotors where 2 diagonal positioned pair of them rotate clockwise and the other diagonal positioned pair rotate counterclockwise. The quadcopter has 6 degree of freedom in 3-D space and each movements are generated by control over the rotational speed of the rotors.

In the vertical plane, quadcopter can rise, descend or hover in the air. All of these moves are provided by the lifting force of the quadcopter rotors where if the generated lifting force which is called as thrust force is more than gravity force, quadcopter moves to upward which is shown in Figure 2.7. On the contrary, if the thrust force is not adequate, quadcopter loses altitude. In the case of balance of the thrust and gravity forces, quadcopter is stabilized at the desired altitude which is named as ”hover”.

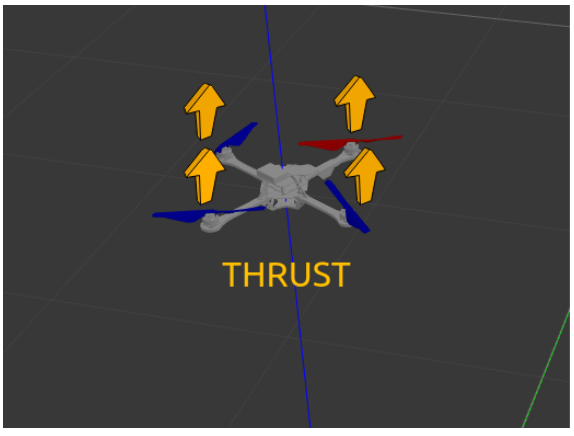

Figure 2.7: Thrust Force.

For the side-way manoeuvre, rotor speeds of the right or the left pairs can be increased according to planned action and the limits of the vehicle. Hence, quadcopter begins to move leftward or rightward as well. Because of the roll,pitch and yaw angle convention adopted, roll angle assumed as positive value for rightward

move and negative value for leftward move. In the Figure 2.8, the leftward move is represented.

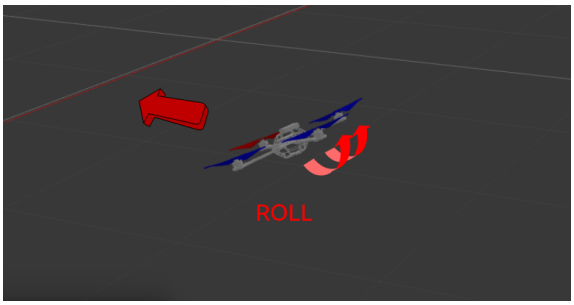

Figure 2.8: Roll to the left.

For the forward move, front pairs rotors speeds decrease and back pairs of the rotors speed increase, however the increase rate of the speed and the decrease rate of speed for the rotors should be equal in order to prevent from losing altitude and imbalanced forces acting on it. Tilt angle which is calling as pitch, is created on the quadcopter while the speed of the rotors are adjusting. As a result, it moves forward or backward respectively. In the Figure 2.9, the pitch move is illustrated.

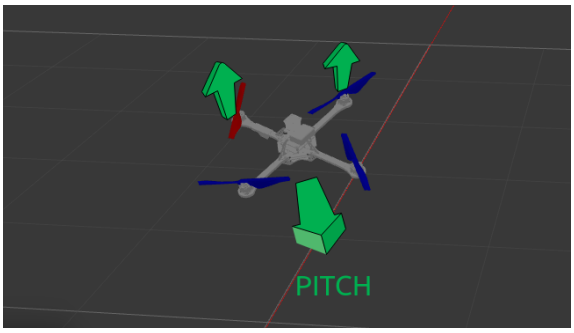

Figure 2.9: Pitch Backward.

When it comes to changing the direction of the quadcopter head, it is easy to handle by adjusting yaw angle. Thanks to angular momentum, if the propellers angular speeds are not increased or decreased relatively, the imbalance of the angular momentum create spin around the origin of the quadcopter. Although the total force and the gravitational force acting on the quadcopter are balanced, quadcopter maintains it's altitude and hovering conditions. The yaw motion is described in Figure 2.10.

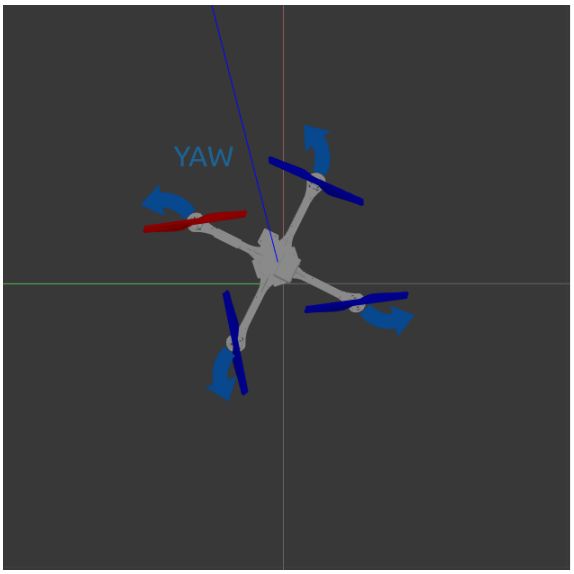

Figure 2.10: Yaw to the left.

2.2.1 Rotation Matrix

It is well known that any rotation of a 3-D point can be defined by using rotation matrix. By using rotation matrix, we can monitor the new rotated linear map of the body respect to the our reference linear map. Since we will use inertial frame and body frame to describe dynamics and motion of the vehicle. For that reason, it is important to explain adopted rotation matrix approach. Roll, pitch and yaw angles are just the kind of Tait-Bryan angles which are also subclass of the Euler angles. The advantage of the Euler angles is not because of their simplicity in calculation and control but also it has intuitively convenient structure to analyse.

Firstly, roll angle is measured through the x axis of the inertial frame as shown in Figure 2.6. The roll angle which is denoted with ϕ symbol. As it is explained before, roll angle takes positive values for counterclockwise changes and negative for clockwise changes. So the roll angle rotation matrix is provided as below:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

Secondly, pitch angle is measured through the y axis of the determined inertial frame also shown in Figure 2.6. Moreover, we used θ symbol to identify pitch angles changes through rotation both in positive for counterclockwise and negative for clockwise changes. Pitch angle rotation matrix is shown as below:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

Thirdly, yaw rotation angle can be identified along the z axis which is also shown in the Figure 2.6. Also, it holds the same counterclockwise positive angle notation

and clockwise negative angle notation like the previous steps. Yaw angle is represented as ψ symbol and the yaw angle rotation matrix is used as below:

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In order to find complete rotation in 3 axis and the new orientation of the body frame, we can multiply firstly the roll matrix and pitch matrix and then yaw rotation matrix respectively. Therefore we can obtain the rotation from inertial frame to body frame. Multiplication order is a important notion which would change the rotations unexpectedly. For the simplicity the cosine function labeled as "c" and the sine function labeled as "s" full rotation matrix from inertial frame to body frame is given as below:

$$R_{IB}(\phi,\theta,\psi) = \begin{bmatrix} c(\psi)c(\theta) & c(\theta)s(\psi) & -s(\theta) \\ c(\psi)s(\phi)s(\theta) - c(\phi)s(\psi) & c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta) & c(\theta)s(\phi) \\ s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) & c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi) & c(\phi)c(\theta) \end{bmatrix}$$

Furthermore, it would be convenient to show the rotation from body frame to inertial frame as well. Since we will use some measurements coming from the quadcopter which is represented as the body frame. So the rotation matrix is shown below:

$$R_{BI}(\phi,\theta,\psi) = \begin{bmatrix} c(\psi)c(\theta) & c(\psi)s(\phi)s(\theta) - c(\phi)s(\psi) & s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) \\ c(\theta)s(\psi) & c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta) & c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\phi)c(\theta) \end{bmatrix}$$

2.2.2 Transformation Matrix to Avoid Heading Angle

As it is briefly explained in the previous section rotation matrix are not only quite significant in order to describe the change in attitude but also for the estimation of the future of the states. In the control approach that we adopted [18], it is suggested that the heading free angle of the drone should be avoided so that heading is not allowed through the change specifically in the attitude. As it is given in the Figure 2.2, the rotation align on the the z axis about the counter clock wise direction can be defined as yaw angle. However, the measured changes in the roll, pitch and yaw angles are defined in the body frame since they are obtained by the change on the their introduced axis such as x, y and z. So that the rotation between the heading free angle which can be assumed as yaw and the roll, pitch angle can be given as follows:

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Because of the adopted control approach and the notation preferences, the control input signal calculation should consider the states which are projected on the inertial frame defined references. The desired command roll and pitch angles are represented with the ϕ_{com} and θ_{com} respectively in the formula. For that reason, the transformation of the roll and pitch angle states from the inertial frame to the body frame by ignoring the heading angle can be formulated as given in the below:

$$\begin{bmatrix} \phi_{bf} \\ \theta_{bf} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \times \begin{bmatrix} \phi_{com} \\ \theta_{com} \end{bmatrix}$$

2.2.3 Forces Acting On Quadcopter

Quadrotor has rigid and x shaped structure which has four rotors vertically oriented and mounted on the each edges. The thrust force which is generated by the each four rotors, is always perpendicular to mounted place which is a edge of the drone[20]. So the generating thrust force can be described as follows [18]:

- F_{th} = Thrust force
- i = Index number of the each rotors
- n_{i} = Angular velocity of the each rotor
- e_z = Unit vector in z direction
- k_n = Positive constant
- k_m = Positive constant

$$F_{th,i} = k_n n_i^2 e_z$$

While the rotors are generating thrust force, it is noticed that they also induced drag force around the rotor structure and that drag force is generating from the angular momentum[20] which is formulated as below [18]:

$$M_i = (-1)^{i-1} k_m F_{th,i}$$

Furthermore, when the quadcopter changes its orientation and position,the angular speed of the quadcopter frame reacts with the gyroscopic torque on it[20]. In addition, the flowing air velocity through the rotor blades induces a differential force between the advancing and receding blades[20]. As it is shown in the [18], explained forces above can be collected into the aerial force equation as shown below:

- $F_{ae,i}$ = Aerial force
- $f_{t,i}$ = z component of the i-th thrust force

- v = Velocity vector of the quadcopter
- k_D = Drag coefficient
- K_{drag} = diag(k_D,k_D,0)

$$F_{ae,i} = f_{t,i} K_{drag} R_{IB}^{T} v$$

2.3 Model Predictive Control

2.3.1 Model Predictive Control Theory

Model predictive control (MPC) has been using and developing in the industrial applications for more than 30 years. The first application fields were mainly related with the process and petrochemical industries. The increased interest of the MPC could be explained by great the performance and the capability. In addition, stability of the control through the process is quite important requirement for the selection of the control approach by the reason of the potential hazardous incidents. On the other hand, the control approach should handle with the more constraints in order to be reliable and robust for the use in the sensitive processes and applications. The model predictive control can offer a solution to the introduced issues, constraints and performance requirements that are commonly faced in the industry. In fact, model predictive control is suitable for the multiple input and output systems(MIMO).

According to the [21] The general objectives of the MPC approach can be listed as a below:

- Generate optimized control signal by taking into account the constraints in both inputs and outputs.
- Force the output variables into their calculated steady state optimal values.
- Force the input variables into the their steady state optimal values.
- Avoid the input variables from the unstable and uncontrollable conditions.
- Maintain the control even if the signal and actuators fail.

In the main sense, the model predictive control differs from the general control approaches thanks to the minimization control signal stage through the objective function and constraints. The procedure of the predictive control approach concept can be described as follows [22]:

- Generate predictions of the future outputs of the system or process in the future time horizon which is defined by the controller designer by considering the accurate and proper mathematical model of the system.

- Calculate the control signals with the minimization of the objective function which relies with the constraints and restrictions.

- Adopt the receding control approach which makes forward time step shifting in the horizon. For the simplicity, receding control approach gives the advantage of being informed about the future of the states which are outputs of the system in the prediction horizon at the current time step. However, only the first calculated and optimized control signal is used and the rest of the calculated control sequence is refused. In addition, this optimization technique is repeated with respect to the recent measurements feedback for the each following time instant in the future horizon .

Moreover, the MPC algorithms can be varied and the regarding dynamic optimization problems can be introduced with the different mathematical model and objective functions modifications through different performance and constraints limitations. Hence, corresponding feature of the MPC offers great freedom for the wide range of the different applications[23]. In addition to that practical convenience, some other advantages of the model predictive control approach which differs from the other classical control methodologies are listed below [22]:

- In the design procedure of the MPC, it allows to handle hard and soft constraints systematically inside the design.

- It is easily applicable for the multivariable input and multivariable output systems.

- In the presence of the measurable disturbances,the sufficient compensation can be achieved thanks to the feed forward control advance in natural way.

- In the case of implementation, the corresponding control law of the MPC is simple to obtain and apply for the controller.

- It is quite effective control approach if a-priori informations about the future actions are exist. For instance, desired and pre-defined trajectory of the robot can be used as the future reference for the trajectory planning applications.

- It can provide satisfactory performance not only limited with the simple dynamics systems but also with the highly complex dynamics system even if the system characteristics involves unstability, non-minumum phase or long delay times.

Besides of the introduced advantages of the MPC, it could be considered as insufficient in a theoretical sense when it overcomes the challenging requirements which are listed as following:

- Even though, the implementation of the controller law is convenient, it may requires more sophisticated derivation rather than common classical approaches.

- Execution time limit of the optimization step is also one of the major obstacle of the MPC implementation. The optimization should be finished as quick as possible inside the each sampling time instant. However, in the case of the higher complexity and the faster dynamics systems with the increased number of constraints may force the optimization to the delay in the time because of the computation time limitations of the processors. On the other hand, this trade off between the performance and the computation time limitation can be adjustable regarding to the proper prediction horizon variable selection which is defined in the designing process of the controller.
- Predictions of the states and outputs are based on the mathematical model which is adopted inside the MPC algorithm directly. For that reason, it is very important to have accurate and sufficient mathematical model which should covers the major dynamics of the system in the MPC design.

It is possible introduce industrial experience of the model predictive control applications since it is commonly used in the petro-chemical and refining industries. Some of the great applications and the different variations of the producers can be listed as in the [22]:

- AspenTech: Dynamic Matrix Control (DMC)
- Adersa: Identification and Command (IDCOM) , Hierarchical Constraint Control (HIECON) and Predictive Functional Control (PFC)
- Honeywell Profimatics: Robust Model Predictive Control Technology (RMPCT) and Predictive Control Technology (PCT)
- Setpoint Inc.: Setpoint Multivariable Control Architecture (SMCA) and IDCOM-M (multivariable)
- Treiber Controls: Optimum Predictive Control (OPC)
- ABB: 3dMPC
- Pavillion Technologies Inc.: Process Perfecter

However, the model predictive control approach is not still extended into the various areas because of the some practical inadequacies which might be related in the case of strong non linearity characters of the objective process and frequently changing operating conditions [22]. According to the [24] some of the industrial practical disadvantages can be listed as follow:

- Overparameterized models: It is commonly known that the impulse response or the step response approach are used to model the process or plant in the many applications of the commercial products and also these type of the models are classified as overparameterized. For example, a first-order process can be used to describe a model by using a transfer function approach which use only three parameters which are gain, time constant and dead time. In the

step response case, it requires at least thirty coefficients in order to describe same dynamics. In addition to that, given models are not suitable and accurate for unstable processes. These problems can be overcome by using an auto-regressive parametric model.

- Tuning: The tuning can not be formulated in the obvious and convenient way because of the changing performance expectations and the changing conditions as well as the changing environment. In addition to that it is not easy to manage the trade-off between the closed loop behaviour and the tuning parameters. Especially, when it comes to the case that constraints are exist, to maintain closed loop stability in the control is not always easy. this is the reason why the MPC designer should make prior simulations. Lastly, the feasibility of the problem is another major obstacle which should be solved in the MPC.

- Suboptimality of the dynamic optimization: There are various types of software which provides suboptimal solutions to the minimization of the cost function to be able to speed up the solution time. Some of the good computation performance convex optimization tools can be used also in the embedded systems which will be showed in the implementation section. It is clear that solution computation time is a very significant issue especially for the high-speed applications like a tracking systems. It should be known that solving the problem at every sampling time may not be feasible, however, it is not easy to justify for process control applications unless it can be shown that the suboptimal solution is always very nearly optimal.

- Model uncertainty: There some model identification packages which are commonly used in the industry in order to estimate the model uncertainty, however, that estimation is used only in the robust model predictive control approach. It is possible to tune the other model predictive controller by managing the trade off between the performance and robustness, although the relation between performance and robustness is not very clear.

- Constant disturbance assumption: Probably one of the most logical approach can be assumed the output disturbance will remain constant in the future, if the distribution of the disturbance could be determined more detailed, it would have better feedback in the control.

- Analysis: In the initial formulation of the finite horizon, there is no systematic way of analysis for the stability and the robustness of the MPC. When it comes to the obtaining control law especially in the constrained case, The control law, in general time-varying and it cannot be represented in the standard closed-loop form. In addition to that the outcomes of the scientific researches and applications in the stability and the robustness manner are limited into quite small area in the sense of state space and control horizon.

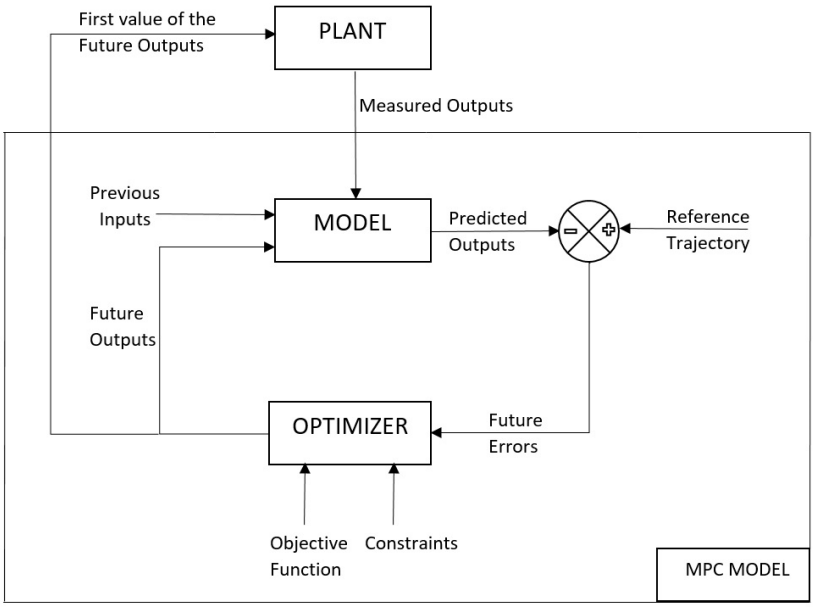

Figure 2.11: MPC Block Diagram

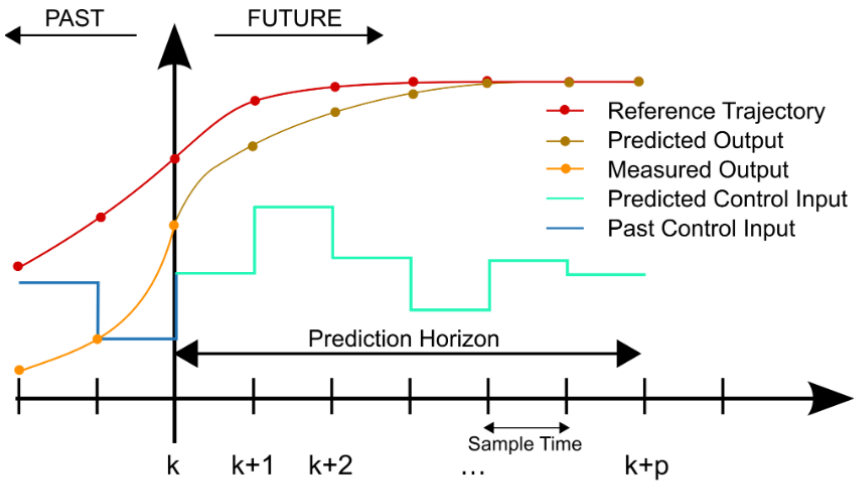

Figure 2.12: MPC Discrete Time Horizon Scheme
By Martin Behrendt

As it is shown in the Figure 2.11, it is possible to visualize the block diagram of the MPC which may improve better interpretation. Future plant outputs are generated from the mathematical model of the plant by using the past and current values of the inputs, outputs and also by considering the optimizer output which is basically the future control signal. This future control signal is generated regarding to the principle of the minimization of the objective function which can be customize as the designer preferences but without violating the initialized constraints which can be both in input and output of the model. It is important to notice that only the first value of the generated future control signal is employed by the mathematical

model and by the plant.

Furthermore, accuracy of the mathematical model of the plant is crucial point for the performance of the proposed control approach. The model should comprise the significant dynamics of the plant, however, the higher complexity of the model extends the computation time which is very critical especially to catch the fast dynamics systems such as the control of the unmanned aerial vehicles.

The performance of the optimizer is also decisive for the MPC. The trade off between the performance, robustness, stability an the trajectory tracking capability can be represented inside the objective function which will be discussed more detailed in the further sections.

It is possible to explain MPC methodology in the discrete time domain horizon as illustrated in the Figure 2.12 and the procedure steps can be described as follows;

- Prediction horizon is the length of the future time instants array which can be adjusted by the performance preferences. As it is given in the Figure 2.8, the prediction horizon length is the number of "p".The use of the notation $x(k+i|k)$ points out that the value of the variable x at the $x(k+i)$ is calculated at the time instant "k". The future output predictions which can be denoted as $y(k+i|k)$ are calculated not only based on the value of the past measured outputs and past control inputs for the time instant index "i" which is starts from "0" to the number of "p" consecutively, but also based on the calculated future control signals $u(k+i|k)$ where the it starts from "1" to the "p-1".

- The calculation of the future control signals $u(k+i|k)$ are based on the optimization preferences which can be named as the minimization of the objective function alternatively. One of the expecting feature of this optimization is to maintain reference trajectory tracking. The performance of the trajectory tracking can be determined as quadratic function which signifies the error between the predicted output signal and the reference trajectory. Also, it is possible to perform constraints in the future control signal prediction.

- The plant only receives the first element of the calculated future control signal which is $u(k|k)$. On the other hand, the rest of the calculated control signal are ignored to send to the plant. Since,at the time "k" ,the output of system for the next time instant which is $y(k+i)$ is already known and the future output predictions are required to updated with this new value again. Hence, the control signal may recalculated respectively.

Daily Life Example

It is possible to demonstrate the practical interpretation of the MPC by giving a simple daily life example. Let's assume that we would like to fill up the bucket with a water but avoiding the overflow in the end. The water flow could be controlled via the water tap to change the water flow rate. Let's imagine that we divide the whole process in to the smaller discrete time steps. Naturally, we can adjust the flow rate relative to the state of the water level and while the water level reaches the to the

top, we gradually reduce the water rate in order to avoid overflow. It is clear that we decide to change the water flow rate according to water level in the bucket and the our instinctive visual prediction. In that step, The control approach can be seen as pid control approach. Because the person regulate the water flow according to the previous setting of the water flow which means the open rate of the water tap. So it can be seen as in the pid approach, previous error is used to update the future action. The major distinction between the pid and the mpc can be noticed in that step. By adopting Mpc approach, if we can predict the future action error which is the possible overflow rate in the example, we can have better control by updating the control signal thanks to the previous error and the predicted future error which can be obtain by considering the reference trajectory. In the mathematical way, let's describe the water filling process by creating proper mathematical model of it and we have only the water flow rate to be controlled. In each time step, water filling rate can be controlled by the prediction of the water level increase and the potential error according the mathematical model and the difference between the prediction and the actual water. Therefore, if we minimize that error by updating the control input which is water flow rate for the next discrete time step, we could optimize the water flow rate in each time steps. Hence, we could have a optimal control which converge the error to the zero subsequently, practically, that avoids the water level from overflow in the end of the process.

2.3.2 Mathematical Models

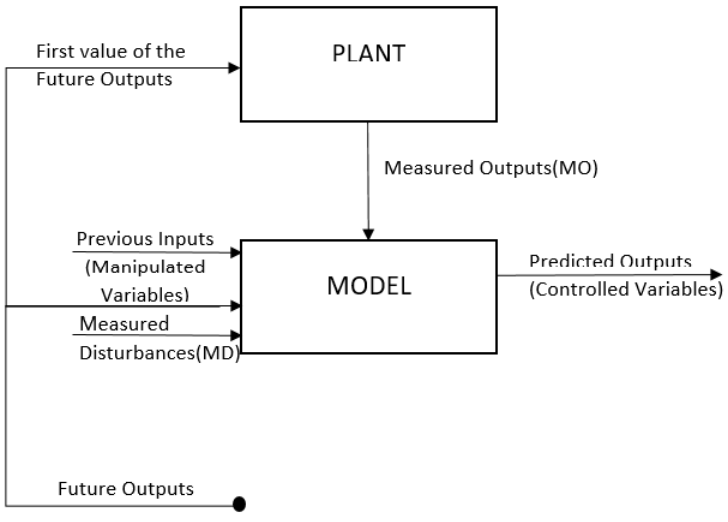

Figure 2.13: Model Blocks of The MPC.

Mathematical Model For Prediction

The mathematical representation of the model of the plant or the system to be controlled is a critical part of the model predictive control design. As it is noted

in the previous parts, the derived mathematical model which is employed for the accurate future prediction of the states should cover the significant dynamics of the plant. However, it should be used in a efficient way such that the complexity of the formulation may remains as low as possible because of the computation cost in the optimization step. As it is shown in the Figure 2.13, manipulated variables which are input signals, the future outputs which are generated by the optimization and the measured outputs which are taken from the plant are executed in the prediction model in order to generate prediction of the outputs.

On the other hand, it is important to consider the unknown disturbances and the modelling errors may affect the accuracy of the prediction. In order to reduce these weakness, model can be modified through the iterations by using state estimator. Unmeasured disturbances and varying parameters of the model could be estimated by using the Kalman filter. However, the use of The Kalman filter could be available only with the linear problems.

Mathematical Model For Plant

Mathematical formulation of the plant is employed to receive the actual measured outputs which is the main goal of the model predictive control application. It is possible to adopt various types of the models by considering the proper and efficient representation of the plant to be controlled. Some of the common used modelling approaches are state space and transfer functions representations which are given as below:

- In the State Space approach, the plant can be represented as a mathematical model in such a way that the first order differential equation or difference equation system. The states can be obtained by the linear arranging of the preceding values of the states itself. The quadcopter dynamics and motions are classed as strong non-linearity nature. Since the equation system constructed as a linear equations, the mathematical representation of the model would be linearized according to the designer target. The linearization for the control of the quadcopter can be applied for the hovering condition. Although, it is important to remember that through the linearization procedure some of the assumptions are made in order to reduce the complexity and replace the nonlinear behaviours with the close linear representations. The general form of the state space representation is given as below:

$$x(k+1) = Ax(k) + Bu(k) \tag{2.1}$$

$$y(k) = Cx(k) \tag{2.2}$$

  To clarify the elements of the system definition, A,B and C are the matrix that signifies the input and output relations of the physical states. Also x is notates the state variables which are selected by the designer. One of the most important convenience of using state space could be that the direct integration and evaluation for the multiple input and output system is possible.

- In the Transfer Function approach, the mathematical model could be derived by executing the laplace transform of the input and output. However, transfer function could be very effective and easy technique, especially, when the system is single input and single output. The output prediction formulation can be given as following:

$$y(t+k|t) = \frac{B(z^{-1})}{A(z^{-1})} u(t+k|t)$$

2.3.3 Objective Function

Objective function can be implemented and customized according to the model predictive control approach and the performance requirement related to the target of the application. The minimization of objective function will generate a control signal array which is obtained by considering the constraints at the input and output, constraints at the control signal effort itself, trajectory tracking performance and some penalties which could be defined for the states. The general form of the objective function could be given as follows [22]:

$$min(J) = \sum_{i=N_L}^{N_H} \delta(i)[\hat{y}(k+i|k) - w(k+i)]^2 + \sum_{i=1}^{N_C} \lambda(i)[\Delta u(k+i-1)]^2$$

As it is shown in the above equation, the length of the prediction horizon which is denoted as "p" could be divided into the two parameters which are N_L for the minimum time instant limit which can be taken as 0 and N_H for the maximum time instant limit which can be determined by the designer. The reason behind this notation is to show that prediction calculation could start at the time instant not only limited for the current time step but also for the selected future time instant period. Additionally, it allows the designer to prevent response of the plant from the non-minimum phase and the sharp response conditions by eliminating the first prediction instants in the case of error thanks to adjusting prediction horizon. N_C is used as the control horizon parameter in the objective function. The "δ" and "λ" terms are used as a coefficients in order to modify and tune the future actions according to the preferences.

The output signal estimation at the time instant k which is shown in the figure 2.8 for the further time instants is defined with the $\hat{y}(k+i|k)$ and also $w(k+i)$ points out the reference trajectory signal which is defined a initially in many applications. Furthermore, the control signal effort is expressed with the $\Delta u(k+i-1)$ term.

Another advantage of the MPC objective function could be given as the integration of the constraints systematically inside the minimization of the objective function. The reason of the usage of the constraints could be explained with the real physical limitations of the components which are performed in the system or plant such as the actuators and sensors. Additionally, these constraints could be used to define the environmental conditions, the energy consumption efficiency and safety regulations. The fact that MPC algorithm may generate optimal solution

even if the variables are approached too close to the constraints. For instance, control signal limitation which is stands for the slew rate and amplitude and the output signal boundary could be added into the objective function which are shown as in the below;

$$u_{min} \leq u(t) \leq u_{max} \tag{2.3}$$
$$y_{min} \leq y(t) \leq y_{max} \tag{2.4}$$

2.3.4 Feasibility

Reliability is a common issue for the proper controller design. In the case of the MPC, computation of the valid control signal which is generated by the optimization should be respected to the given constraints. However,for the some challenging conditions, optimization may not provide sufficient solution which satisfies for the all given constraints. Therefore, it creates an in-feasibility problem which can be related also with the strong disturbances. In the initialization of the mpc, it may be far from the desired operating range and even it may violate some of the constraints. When MPC faced with a such a in-feasible optimization problem, it is significant that the controller should maintain the solution and avoid terminate the operation. Instead, it is expected to observe decrease in the performance of the controller respect to the increase of the violation of the constraint and also controller should be able to drive the process into an operation region where all constraints are feasible.

In the case of the incoherent constraints are applied for the optimization, if there is no operating point exist for the optimization than the problem formulation should be modified. Coherence of the constraints can be ensured through practical point of view where the physical limits of the components of the system to be controlled are known. In addition to that, physical constraints cannot be violated. The constraints which are directly effects the input signal should not be violated by the optimized solution. In contrast, states and outputs constraints do not always signify the fundamental operational restrictions, sometimes they indicate only for the preferable operational conditions. So that violation on the state or output constraints is a possible option for a brief time interval. For this purpose, MPC optimization problem can be modified in order to allow the constraints violation on the state and output. Some of the possible modification strategies can be implemented as a given below;

- Eliminating the state and output constraints for the a close range time interval. However, it should be known that it may result with excessive amount of a redundant constraints violation. Also determining the duration of the required time interval for the constraints elimination may not be easy to find. Since this modification should be done by considering the input constraints, operation range and the severity of the disturbance.
- Solving the infeasible part of the optimization problem by separating before the actual optimization process. In this concept, it is important to determine how many state and output constraints should be violated in order to have feasible

optimization problem. Also if the separate computed part of the optimization problem can be defined as linear programming problem, the solution can be obtained conveniently.

- Another approach can be adopted by defining the penalty function inside the optimization problem. Constraints can be modified in order to maintain feasible solution by proposing additional variables which are adequately large values. This modification converts the hard constraints into the labelled as soft constraints. Naturally, objective function should be replaced with the new modification of the constraints by adding a term which provides the penalty on the magnitude of the constraints violations. These located variables provide the feasibility of the constraints and converted into free variables in the optimization. Even the feasibility is maintained, however, the size of the problem is already increased.

2.3.5 Stability

Thanks to the model predictive control optimization approach, it is naturally expected to ensure stability in a closed-loop control system. In the most of the model predictive control implementations, receding horizon strategy is adopted, so that successfully developed optimization up to the horizon can be seen as pretty much optimal value. However, it may not fully satisfied with some requirements. One of the suggested approach according to this situation may be used in the case of the positive definite cost function exist. The cost function can be considered as Lyapunov function and the stability analysis may become efficient[25].

On the other hand, there is still another difficulty exist because of the optimization problem which is defined only up to the finite optimization horizon. For that reason, it is possible to say that about the overall stability in period of the finite horizon is not a complicated issue since finite horizon which is greater than the optimization horizon. A possible convenience could be add a proper cost function weighting and constraints on the terminal state of the optimization horizon in order to notice the impact of the actions which are directly related with the optimization horizon[26]. So that system should be driven into the given restricted terminal region for a time which is determined from the optimization calculation time availability. From this point of approach, stability can be ensured by the system manipulation. It is possible to have further information about the Lyapunov function which is employed for the cost function in the [27] and also the manipulation of the terminal constraints in order to maintain stability in the [28].

Chapter 3

Quadcopter Control and Trajectory Planner

3.1 Introduction

In this chapter we will briefly explain the control approach which is adopted.

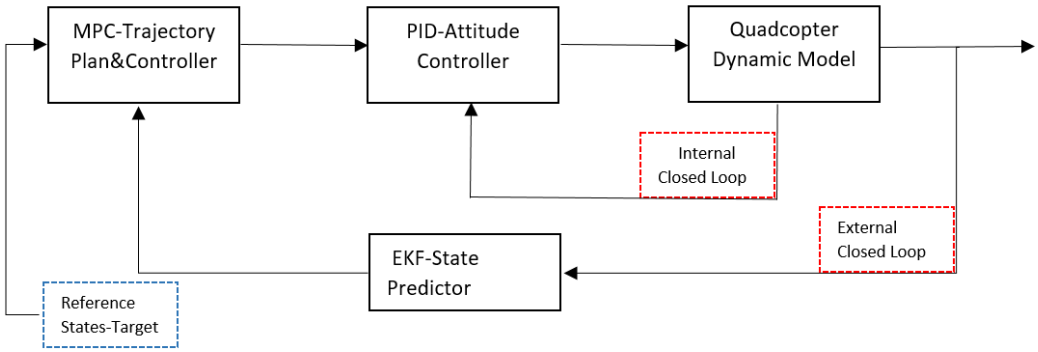

Figure 3.1: Quadcopter Control Block Diagram.

Typically, quadcopter system can be seen as nonlinear and unstable in terms of the control. For that reason, it is crucial to design and select proper tools such as sensor and the control approach. According to the [14],in that sense, there are techniques that have been developed to solve this problem. A common strategy which is employed from the researchers is to develop cascaded control approach. They suggested to divide the complex control system in to the two connected subsystems. Therefore, one of the divided subsystem can be employed for the attitude purpose and the other one can be used for the positional control over the coordinates. Particularly, the attitude control problem has been largely examined and many possible solutions have been found. For instance, in the [17], the researchers determined the advantages and the disadvantages of the "Backstepping" and the "Sliding Mode" controller approaches. Both these methods provide a satisfactory

stabilize solution to the control problem. However, according to the [14], This is performance and stability problem for the attitude controller can be tackled by using simple PID controller as well. In the Figure 3.1, quadcopter control block diagram is visualized in order to have better interpretation. The adopted approach could be investigated and the input output relations can be seen in the general sense. In the further section more detailed information about the controller and the block will be introduced. As it is given in the Figure 3.1, the approach is divided in to two subsystem which are internal closed loo and the external closed loop. In the internal closed loop, attitude requirements are controlled and regulated. On the other hand, in the external closed loop trajectory tracking and the positional requirements are handled. It is important to note that the transition between the internal and the external loop will be examined in the further sections.

3.2 Internal Control Loop

In this chapter we would like to introduce the adopted internal loop control approach. One of the major issue arises in the cascaded control approach is to paired the internal and the external subsystems properly in order to have good trajectory tracking ability and the good attitude control. For that reason, external control method might be designed to be react to the changes at the attitude for the calculated time.

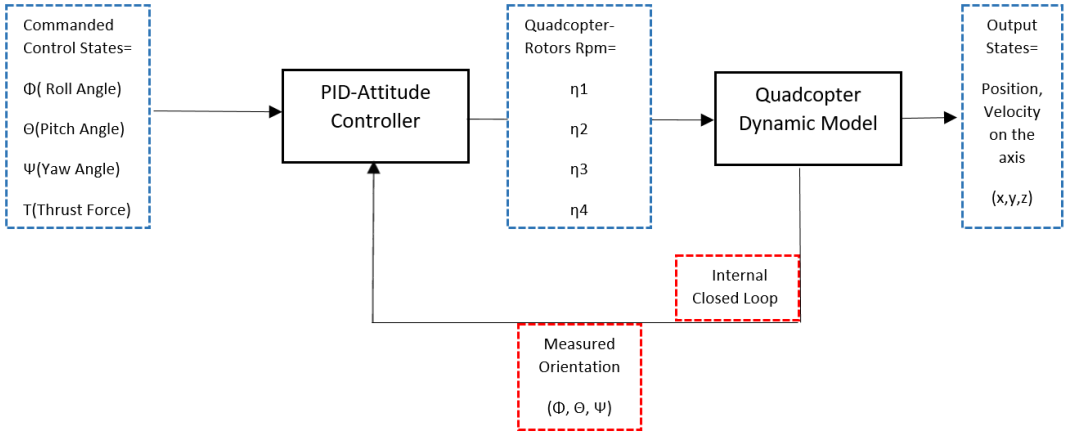

Figure 3.2: Quadcopter Internal Control Block Diagram.

3.2.1 Quadcopter Dynamic Model

Mathematical modelling of the quadcopter is quite critical process in order to describe quadcopter moves and response to given input selection. Intuitively, the relationship between the input and the output of the system can be determined

with the functions which provide the characteristic of the quadcopter. Additionally, it is possible to predict and simulate the future move, states and the response for the changing conditions of the environment by controlling the rotor speed of the propellers which are mounted on the quadcopter[29]. In order to improve accuracy and reduce the uncertainty of the model of quadcopter, more complex and detailed modelling can be chosen. However, it would have high computational cost for the processors in real the applications. Hence, this trade off between the accuracy and the practicability could be overcome by considering only the important dynamics of the quadcopter and by neglecting the minor priority dynamics in the mathematical model[29].

Some of the assumptions are adopted through the obtaining dynamical mathematical model as follows:

- Centre of mass of the quadrotor positioned at origin of the body fixed frame.
- Axes of the body frame matches up with the body principal axes of inertia.

In this section, we explain the dynamical modelling approach which we adopted in this work. The given equations hold Newton's law of motion as well.And we followed the modelling approach which is given in the [18]. Also it is convenient to define used symbols as a given below:

ρ = Position of the center of the gravity of the quadcopter reference to inertial frame

υ = Quadcopter velocity reference to inertial frame

F_{th} = Thrust force which is generated from the rotors as a lifting force

F_{ae} = Aerodynamics force

F_{ex} = The external forces

R_{IB} = Quadcopter orientation

ω = Quadcopter body angular rate

m = Quadcopter mass

J = Inertia matrix

A = Allocation matrix

g = Acceleration due to gravity

n_i = Rotor speed

N_r = Number of the rotors

Consecutively the motion of the quadcopter could be defined [18]as a written below;

$$\dot{\rho} = \upsilon$$

$$\dot{v} = \frac{1}{m}(R_{IB}\sum_{n=0}^{Nr} F_{th,n} - R_{IB}\sum_{n=0}^{Nr}(F_{ae,n} + F_{ex}) + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

$$\dot{R_{IB}} = R_{IB}[\omega\times]$$

$$J\dot{\omega} = A\begin{bmatrix} n_1^2 \\ n_2^2 \\ n_3^2 \\ n_4^2 \end{bmatrix} - \omega \times J\omega$$

3.2.2 Quadcopter Dynamic Model Linearization

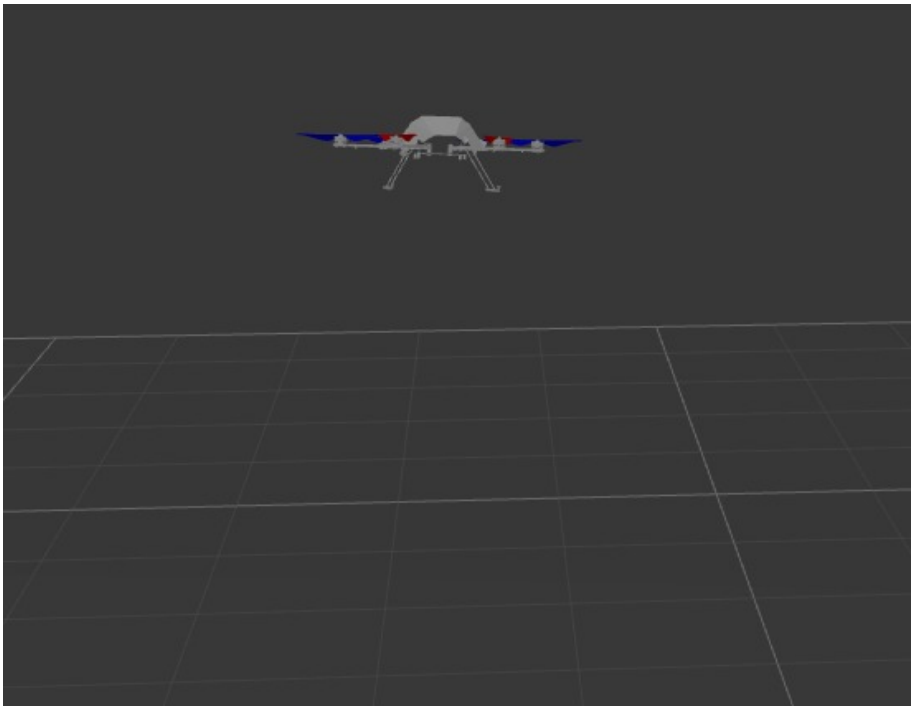

Figure 3.3: Quadcopter Hovering Condition.

As it is explained in the previous section, the quadcopter dynamics have some strong nonlinearities because of the mathematical modelling which is referenced by it's nature. For that reason, mathematical model might be linearized in order to make it available for the linear control approaches and for the simplification of the nonlinear system. In addition to that, linearization could be applied around the equilibrium points which are basically the derived solution of the system states where the derivatives of the states are taken as zero condition. In the proposed linearization case, linearization is applied for the hovering condition of the quadcopter. The hovering condition can be described in a such a way that, the quadcopter stays at desired altitude without any orientation and altitude changes. The visualization of the hovering condition can be seen in the Figure 3.3. The linearization around

the stable quadrotor hovering condition is quite critical step in order to have proper control law which is capable to control the quadcopter system and maintain trajectory tracking[30]. Moreover, the adopted linearization approach neglects the small orientation angles changes and the yaw angle "ψ" considered as zero.

3.2.3 Internal Measurement Unit

Internal Measurement Unit (IMU) is a sensor that consist of the 3-axis accelerometer and 3-axis gyroscope inside. It can be employed to measure the angular rates, position and force. Because of the sensor composition inside, it operates the measurement through the 6-axis which makes 6 degree of freedom which can be seen in the Figure 3.4. The accelerometer is used to determine the linear acceleration through a single axis and the direction. In addition to that it is can be performed in order to measure gravity force and for the position. When it comes to measure angular velocity changes, the accelerometer may not be sufficient enough. From this standpoint, Gyroscopes, can be considered as solution for this specific problem. Each angular velocities of the principal three axis which are orientation angles are roll angle "ϕ", pitch angle "θ" and the yaw angle "ψ" can be found directly thanks to the gyroscope sensor inside the IMU.

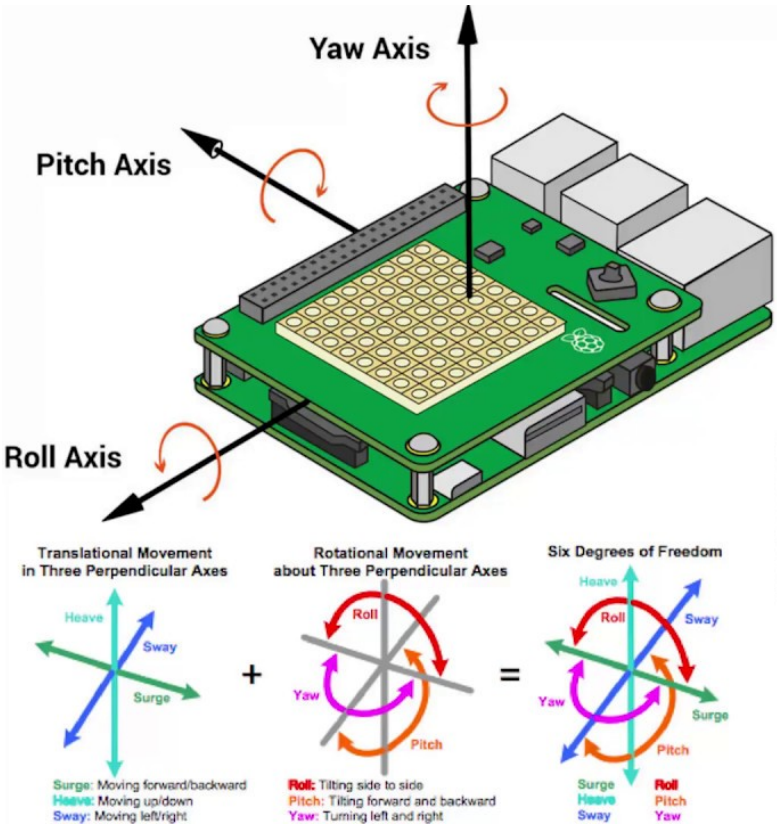

Figure 3.4: Internal Measurement Unit.

3.2.4 PID Attitude Controller

As it is given in the Figure 3.2, commanded control states which are generated from the linear model predictive controller (LMPC) is taken as a input for the PID-Attitude controller. The PID attitude controller considers not only the input coming from the LMPC but also the input which is the measured orientation angles. The orientation measurement is done thanks to the internal measurement unit. Practically, these orientation angles are roll angle "ϕ", pitch angle "θ" and the yaw angle "ψ". The practical demonstrations of the orientation angle are done in the Notation section. Moreover, Attitude controller is designed to regulate the roll angle and the pitch angle. Also the the yaw angle is defined in a such a way that the rotation around the vertical axis "z" as a angular velocity of the quadcopter. In addition to that total thrust force is also determined by the attitude controller. As it can be seen in the Figure 4.2, PID- attitude controller generates appropriate rotor signal to the four rotor of the quadcopter in order to maintain calculated orientation. The attitude controller could be identified by applying basic system identification techniques in the case of unknown controllers exist on the commercial quadcopter systems. According to the [18], the attitude controller dynamics for the roll move and angle is given as follows:

$$\dot{\phi} = \frac{k_\phi \phi_{com} - \phi}{\tau_\phi}$$

The term "$\dot{\phi}$" signifies the roll angle rate and the "k_ϕ" is denoted for the determined gain and the "τ_ϕ" is denoted for the determined time constant thanks to the system identification techniques. The "ϕ_{com}" term is used to remark commanded roll angle. The attitude controller dynamics for the pitch move and angle is given as follows:

$$\dot{\theta} = \frac{k_\theta \theta_{com} - \theta}{\tau_\theta}$$

As it is given in the previous equations,The term "$\dot{\theta}$" points out the pitch angle rate "k_θ" is specified by the determined gain, also the "τ_θ" is used for the obtained time constant and the "θ_{com}" is employed for the commanded pitch angle. It is significant to notice that the yaw angle rate might be taken directly as equal to the commanded yaw angle rate since the quadcopter is expected to react the command reference so quickly according to the our assumption. So it can be formulated as follows;

$$\dot{\psi} = \psi_{com}$$

3.3 External Control Loop

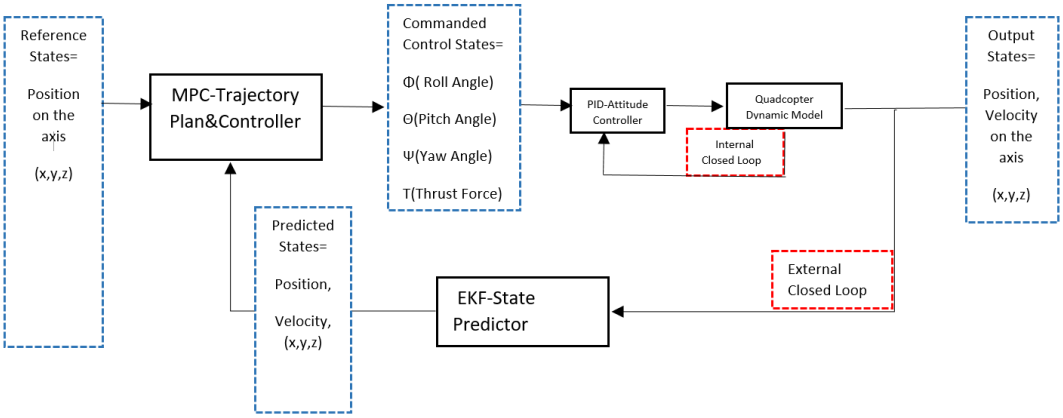

Figure 3.5: Quadcopter External Control Block Diagram.

In this section we will introduce external control loop in details. As it can be seen in the Figure 3.5, external control loop comprises linear model predictive control which is dedicated for the trajectory planning and the trajectory tracking capabilities and the extended Kalman filter which is employed for the proper and accurate state estimation to feed up the (LMPC) controller. The controller takes two kind of input as it is seen in the Figure 3.5, the reference states are designed and introduced from the user as a target for the quadcopter. So that, quadcopter may be forced to have or to be close to the given reference values of the states. The measured output states are considered by the state predictor in order to feed the controller as well. Further details about the controller and the extended Kalman filter can be found in the next sections.

3.3.1 Linear Model Predictive Control

Linear Model Predictive Control can be seen as optimal control which aims to obtain a control law that considers the series of differential equations relied on the constraints and reduces cost by optimizing the state variables in the defined cost function[31]. Variables states are determined as follows:

$$x_{states} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ v_x \\ v_y \\ v_z \\ \phi_{in} \\ \theta_{in} \end{bmatrix}$$

The position is defined with the P on the x,y and z axis in the inertial frame as it is shown above and the linear velocities which are also defined with the v on the x,y

and z axis in the same frame. So these positional and velocity states are measured by the internal measurement unit which is described in the previous section. The roll angle variable which is defined as "ϕ_{in}" and the pitch angle variable "θ_{in}" are obtained thanks to the body frame to inertial frame transformation which avoids the use of yaw angle in the model. Since the roll and the pitch angles are measured in the body frame with internal measurement unit as well. The input variables are defined as given in the below:

$$u_{input} = \begin{bmatrix} \phi_{com} \\ \theta_{com} \\ T_{in} \end{bmatrix}$$

The "ϕ_{com}" can be used to describe the input commanded roll angle and the "θ_{com}" is used to define the input commanded pitch angle. The "T_{in}" variable is designated for the thrust force and determined in the inertial frame[18].

Furthermore, according to the quadcopter dynamic model linearization section, it is possible to introduce vehicle dynamics as shown in the below in order to complete controller design:

$$\dot{x} = A_{ct}x(t) + B_{ct}u(t) + B_{d,ct}d(t)$$

Also it can be convenient to give the extended matrix representation of the initialized quadcopter dynamic equation system in order have better interpretation as follows:

$$\dot{x} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -A_x & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & -A_y & 0 & 0 & -g \\ 0 & 0 & 0 & 0 & 0 & -A_z & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} \end{bmatrix} x(t) +$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{K_\phi}{\tau_\phi} & 0 & 0 \\ 0 & \frac{K_\theta}{\tau_\theta} & 0 \end{bmatrix} u(t) +$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} d(t)$$

However, the quadcopter dynamic system is represented as continuous time domain as it can be seen in the above, it is necessary to convert continuous time domain representation into the discrete time domain in order to make suitable for the controller which is defined in the discrete time domain[32]. For that purpose, it is possible to transform the the matrix respecting to the predefined sampling time "T_s" such that;

$$A_{(dt)} = e^{A_{(ct)}T_s} \tag{3.1}$$

$$B_{(dt)} = \int_0^{Ts} e^{A_{(ct)}\tau} d\tau B_{(ct)} \tag{3.2}$$

$$Bd_{(dt)} = \int_0^{Ts} e^{A_{(ct)}\tau} d\tau Bd_{(ct)} \tag{3.3}$$

In the next step, the control input might be calculated by adding feed-forward term to preserve from the coupling issues. In addition to that, quadcopter control signal should consider the initial upward thrust force which performs against to the gravitational force and air drag in order to maintain the quadcopter in the hovering condition where the stabilization and linearization are executed. Similarly, the good trajectory tracking capability might be maintained and the control inputs can be introduced as given in the below:

$$\tilde{T} = \frac{T + g}{\cos(\phi)\cos(\theta) + \ddot{z}_d} \tag{3.4}$$

$$\tilde{\phi_d} = \frac{g\phi_d - \ddot{y}_d}{\tilde{T}} \tag{3.5}$$

$$\tilde{\theta_d} = \frac{g_d + \ddot{x}_d}{\tilde{T}} \tag{3.6}$$

According the given equations at the above, the terms "$\tilde{T}$", "$\tilde{\phi_d}$" and the "$\tilde{\theta_d}$" are employed to address actual control inputs. The "g" refers to the gravitational force acting on the quadcopter. Moreover, the feed-forward terms "$\ddot{x}_d$", "$\ddot{y}_d$" and the "$\ddot{z}_d$" are initialized to point out the desired acceleration of the quadcopter on the three dimensional space directions respect to the defined body frame.

3.3.2 Extended Kalman Filter State Prediction

The disturbances may be modelled and considered in the discrete time mathematical model of the system in order to maintain trajectory tracking capabilities as we desired to have according to the objective. For the convenience term "d_k" can be adopted as a modelled disturbance to describe system model inconsistency. So the system output can be introduced as follows:

$$y_k = Cx_k$$

In addition to that, we would like to have convergence characteristic of the error between the estimation states and the actual states for the future of the system. For that reason it is possible to define the "$\hat{x}$" for the estimated states and the "$\hat{d}$" for the estimated external disturbance which can be interpreted as strong wind applied on the quadcopter. "L_x", "L_d" are the gains which are computed for the observer and the "y_{mk}" may be determined as a output in the discrete time as it is given in the [32]. According to the given approach the Extended Kalman Filter can be performed to have proper estimation of the disturbance and the states by referring the model which is performed as given in the below:

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{d}_{k+1} \end{bmatrix} = \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} U_k + \begin{bmatrix} L_x \\ L_d \end{bmatrix} (C\hat{x}_k - y_{mk}) \tag{3.7}$$

In addition to that, in order to maintain stable observer for the state and disturbance predictions to feed the linear model predictive controller, it may required to determine state reference and the control input reference as for the discrete time in the equation below;

$$\begin{bmatrix} A - I & B_{d,k} \\ C & 0 \end{bmatrix} \begin{bmatrix} x_{ref,k} \\ u_{ref,k} \end{bmatrix} = \begin{bmatrix} -B\hat{d}_k \\ r_k \end{bmatrix}$$

(3.8)

3.3.3 Linear Model Predictive Control Objective Function

The general structure and the prior knowledge about the objective function is given in the previous sections. Furthermore, it is convenient to propose the adopted quadratic form minimization of the objective function as follows:

$$J(u,x) = \min\Bigg[\Bigg(\sum_{i=0}^{N-1}(x_{k+i} - x_{ref,k})^{\mathsf{T}} Q_x (x_{k+i} - x_{ref,k}) + (u_{k+i} - u_{ref,k})^{\mathsf{T}} R_u (u_{k+i} - u_{ref,k}) + (u_{k+i} - u_{k+i-1})^{\mathsf{T}} R_\Delta (u_{k+i} - u_{k+i-1})\Bigg) + (x_N - x_{ref,N})^{\mathsf{T}} P (x_N - x_{ref,N})\Bigg]$$

subject to :

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + Bd_k; \\ d_{k+1} &= d_k; \\ u_k &\in U; \\ x_0 &= x_{t0}; \\ d_0 &= d_{t0} \end{aligned} \tag{3.9}$$

As it is given in the the below, the optimal control problem is defined according to the linearized system dynamics and respect to the linear constraints which can be declared both for the input, output and the states. Additionally, the efficient trajectory tracking capability can be expected as one of the major requirement for the model predictive control and this requirement has been broadly studied and many reasonable solutions have been implemented both for the linear and nonlinear model predictive control [33]. To overcome given issue, one of the convenient way might be followed the [34], the disturbance can be introduced as a steady state "$d(t)$" and the model of the linearized system can be upgraded with the disturbance state in order to compensate the model inaccuracy which is caused by the system model uncertainties.

Moreover, the "x_{ref}" , "u_{ref}" terms are used to address the reference signal which might be altered according to the targets. The term "Bd" is employed to introduce the disturbance model that could be designed respecting to the disturbance form and "$d(k)$" represents the external disturbance as well.

Further, the term "P" is used to define the penalty on the terminal state error[32]. The weighting matrices Q_x, R_u and R_Δ are initialized as positive semi-definite and they signify the penalties for the errors which are respectively on the states, control input and the control change rate. It is worth noting that the variation in the control input magnitude is restricted by enforcing the linear constraints through the objective function.

When it comes into the stability and feasibility of the mentioned linear model predictive controller, it may not be convenient to obtain a specific strategy that has been proposed for this problem. A possible approach to solve this problem suggests to use longer prediction horizon which is defined as "N" in our representation, however, it may not guaranteed the stability and feasibility of the controller[35]. In addition to that, extending the prediction horizon length may rise the complexity of the minimization problem which would cause to the computation burden. As it is emphasized in the preceding sections, the control input calculation should be fast

enough to apply to the quadcopter system since the dynamics can be seen as quite agile. On the other hand, another approach which is mentioned in the [36] confirms that the terminal constraint "P" adjusting can be good method for maintaining closed-loop stability and the feasibility.

3.4 Robotic Operating System Implementation

The defined quadcopter dynamics and the trajectory planning actions are evaluated and integrated in the ROS environment which is introduced in the previous sections. Moreover, the programming language is selected as C++ because of the advantage which allows us to manage the memory allocation and create a way that we can interpret the algorithm as in the machine code level. The use of The ROS environment can offer great number of packages which are already developed by the other developers. These packages are very useful especially in order to avoid to write common use tools and algorithms again and again. The other advance can be introduced through the varying code language adaptation. For instance, it allows to develop a script in the python language and integrate in to the C++ packages without any cost. In addition to that, it is possible to simulate the defined systems in the common simulation environment which is "Gazebo". So the real world simulations are available because of this convenience.

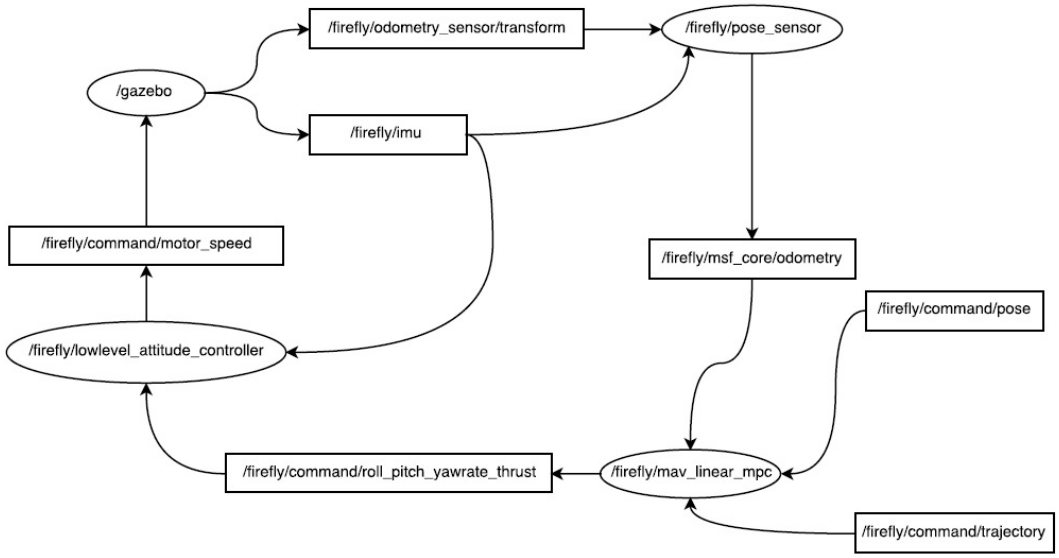

Figure 3.6: Active Ros Nodes.

In the Figure 3.6, It is possible to demonstrate actively employed ROS nodes which performs required communications between the systems [18]. The created linear model predictive controller script is developed as a node which is named as "mav_linear_mpc". This node receives 3 different messages from the 3 different nodes. In more terminological way, The "mav_linear_mpc" node subscribes the odometry node which is generating odometry message which contains state variables measured by the sensor system and describes the actual odometrical variables. The

"command_pose" is defined by the user in order to define target position for the quadcopter. And the trajectory message is created thanks to the optimization step which is covered by the convex optimization solver. Therefore, the "mav_linear_mpc" node publishes the calculated roll, pitch and the yaw rate to the internal loop unit which is PID controller named as "lowlevel_attitude_controller" node. As it is described in the internal loop section, PID controller calculates related required motor speeds as a revolution per time unit for the each rotors. It is significant to point that, internal control loop which is operated by PID performs at 100 Hz because of the attitude dynamics, however, outer loop which is operated by the linear model predictive controller can perform slower than inner loop.

3.4.1 Trajectory Planning With CVXGEN

For the trajectory tracking path planning system we followed the "CVXGEN" solver which is offered as in the [37]. In the embedded system applications, it is important to use robust and accurate convex optimization solver since the solution should be available by avoiding any fatal error. On the other hand, a computation time is well-known problem particularly for the UAV embedded system applications. Also, we should remind that computation time takes into account the complexity of the quadratic optimization problem. It is commonly known that the computation time problem could not be easily tackled by reducing the complexity of the problem since the system model should maintain significant dynamics of the actual quadcopter system. The efficient computation time can be obtained by using "CVXGEN" solver. According to the [37], it is possible to show the computation time performance and the complexity allowance of the solver in the Figure 3.7. After the proper installation of the system dynamics, constraints

	Size (m, n)		
	Small (3, 10)	Medium (6, 20)	Large (12, 40)
CVX and SeDuMi	230 ms	260 ms	340 ms
Scalar parameters	143	546	2132
Variables, original	10	20	40
Variables, transformed	10	20	40
Equalities, transformed	3	6	12
Inequalities, transformed	20	40	80
KKT matrix dimension	53	106	212
KKT matrix nonzeros	165	490	1620
KKT factor fill-in	1.00	1.00	1.00
Code size	123 kB	377 kB	1891 kB
Generation time, i7	0.6 s	5.6 s	95 s
Compilation time, i7	1.1 s	4.2 s	56 s
Binary size, i7	67 kB	231 kB	1256 kB
CVXGEN, i7	26 µs	110 µs	720 µs
CVXGEN, Atom	250 µs	860 µs	4.6 ms
Maximum iterations required, 99.9%	8	9	11
Maximum iterations required, all	20	17	12

Figure 3.7: CVXGEN Performance
[37]

It is convenient to introduce how the linear model predictive controller parameters are used and applied into the ”CVXGEN” solver in that step. For that purpose, we can start to introduce how the model of the system is created and how the parameters are assigned for the solver in the ”mav_linear_mpc” script. The model continuous time model can be created as follows;

Code Fragment

```
Eigen::MatrixXd A_continous_time(kStateSize, kStateSize);
A_continous_time = Eigen::MatrixXd::Zero(kStateSize,
   kStateSize);
Eigen::MatrixXd B_continous_time;
B_continous_time = Eigen::MatrixXd::Zero(kStateSize,
   kInputSize);
Eigen::MatrixXd Bd_continous_time;
Bd_continous_time = Eigen::MatrixXd::Zero(kStateSize,
   kDisturbanceSize);
A_continous_time(0, 3) = 1;
A_continous_time(1, 4) = 1;
A_continous_time(2, 5) = 1;
A_continous_time(3, 3) = -drag_coefficients.at(0);
A_continous_time(3, 7) = kGravity;
A_continous_time(4, 4) = -drag_coefficients.at(1);
A_continous_time(4, 6) = -kGravity;
A_continous_time(5, 5) = -drag_coefficients.at(2);
A_continous_time(6, 6) = -1.0 / roll_time_constant_;
A_continous_time(7, 7) = -1.0 / pitch_time_constant_;
```

Code Fragment

```
B_continous_time(5, 2) = 1.0;
B_continous_time(6, 0) = roll_gain_ / roll_time_constant_;
B_continous_time(7, 1) = pitch_gain_ /
   pitch_time_constant_;

Bd_continous_time(3, 0) = 1.0;
Bd_continous_time(4, 1) = 1.0;
Bd_continous_time(5, 2) = 1.0;
```

After that, it is possible to convert the model in the discrete time domain for the specified sampling time of the system. The discrete time model of the system is evaluated to use not only for the steady state calculations but also for the parameters in the " CVXGEN ".

Code Fragment

```
model_A_ = (prediction_sampling_time_ *
   A_continous_time).exp();

Eigen::MatrixXd integral_exp_A;
integral_exp_A = Eigen::MatrixXd::Zero(kStateSize,
   kStateSize);
const int count_integral_A = 100;

for (int i = 0; i < count_integral_A; i++)
{
  integral_exp_A += (A_continous_time *
     prediction_sampling_time_ * i /
     count_integral_A).exp()
      * prediction_sampling_time_ / count_integral_A;
 }

 model_B_ = integral_exp_A * B_continous_time;
 model_Bd_ = integral_exp_A * Bd_continous_time;
```

As we obtained the model, it is possible to assign parameters of the solver such as;

Code Fragment

```
Eigen::Map<Eigen::MatrixXd>(const_cast<double*>(params.A),
   kStateSize, kStateSize) = model_A_;
Eigen::Map<Eigen::MatrixXd>(const_cast<double*>(params.B),
   kStateSize, kInputSize) = model_B_;
Eigen::Map<Eigen::MatrixXd>(const_cast<double*>(params.Bd),
   kStateSize, kDisturbanceSize) =
    model_Bd_;
```

In addition to that, the other required parameter variables are calculated in varying proper methods, however, it is not favorable to show all the parameters which are used by solver in this section because of the extensive size of the code. Some of the parameters can be seen in the below ;

Code Fragment

```
Eigen::Map<Eigen::MatrixXd>(const_cast<double*>
(params.Q), kStateSize, kStateSize) = Q;
Eigen::Map<Eigen::MatrixXd>(const_cast<double*>
(params.Q_final), kStateSize, kStateSize) =
    Q_final;
Eigen::Map<Eigen::MatrixXd>(const_cast<double*>
(params.R), kInputSize, kInputSize) = R;
Eigen::Map<Eigen::MatrixXd>(const_cast<double*>
(params.R_omega), kInputSize, kInputSize) = R_delta
    * (1.0 / sampling_time_ * sampling_time_);

params.u_max[0] = roll_limit_;
params.u_max[1] = pitch_limit_;
params.u_max[2] = thrust_max_;

params.u_min[0] = -roll_limit_;
params.u_min[1] = -pitch_limit_;
params.u_min[2] = thrust_min_;
```

After the proper and accurate initialization of the parameters and the constraints of the system, it is convenient to obtain optimal control input command by calling the solve function which invokes the "CVXGEN" and we may count the solution time by using "tic() and toc()" in order to guarantee the compatibility with the system frequency. The optimum control inputs are stored in the "linearized_command_roll_pitch_thrust_ " The following code scripts is adopted in the algorithm such that;

Code Fragment

```
tic();
int solver_status = solve();
solve_time_average_ += tocq();

linearized_command_roll_pitch_thrust_ << vars.u_0[0],
    vars.u_0[1], vars.u_0[2];
```

In the Figure 3.8, it is convenient to present how to initialize the objective function optimization problem into the "CVXGEN" interface. Firstly, proper dimensions of the matrix are defined, then the prediction horizon which is addressed as "T" is defined according to the preferences. After that step, the parameters dimensions are defined in the parameter section, however, they are initialized and calculated in the "mav_linear_mpc" node. For the variables section, modelled system states and inputs are defined. The objective function of the linear model predictive control algorithm is derived to be used in the minimization respected to the "CVXGEN"

interface. After that, constraints are defined according to the performance requirements and physical limitations. For instance, control input limits are expressed as "u_min" and "u_max". The exporting C program files are automatically generating by the interface and ready to integrate into the project files. However, for the obstacle avoidance case, it may not be possible to adjust the "CVXGEN" problem definition and initialization scheme in order to implement mixed linear programming techniques. In addition to that, the Boolean type of variable is not supported in the constraints definition section in the "CVXGEN" interface. For that reason, another convex optimization solver which is "CVXPY" is adopted in order to maintain obstacle avoidance.

```
dimensions
  m = 3 # dimensionofinputs.
  nd = 3 # dimensionofdisturbances.
  nx = 8 # dimensionofstatevector.
  T = 20 # horizon-1.
  k = 4
end

parameters
  A ( nx , nx ) # dynamics matrix.
  B ( nx ,m) # transfermatrix.
  Bd ( nx , nd ) # disturbancetransfermatrix
  Q ( nx , nx ) psd # statecost,positivesemidifined.
  Q_final ( nx , nx ) psd # finalstatepenalty,positivesemidifined.
  R (m,m) psd # inputpenalty,positivesemidifined.
  R_omega (m,m) psd # deltainputpenalty,positivesemidifined.
  x[0] (nx) # initialstate.
  d ( nd ) # disturbances.
  u_prev (m) # previousinputappliedtothesystem.
  u_max (m) # inputamplitudelimit.
  u_min (m) # inputamplitudelimit.
  x_max
  x_min
  y_min
  y_max
  x_ss[t] (nx), t=0..T+1 # referencestate.
  u_ss (m)  # referenceinput.

end

variables
  x[t] ( nx ) , t=1..T+1 # state.
  u[t] (m) , t=0..T # input.

end

minimize
  quad(x[0]- x_ss[0], Q)+ quad(u[0]-u_ss, R) + quad(u[0]-u_prev, R_omega) + sum[t=1..T](quad(x[t]-x_ss[t], Q)+
  quad(u[t]-u_ss, R) + quad(u[t] - u[t-1], R_omega)) + quad(x[T+1]-x_ss[T+1], Q_final)
subject to
  x[t+1]==A*x[t]+B*u[t]+Bd*d, t=0..T
  u_min<=u[t]<=u_max, t=0..T

end
```

Figure 3.8: CVXGEN Interface.

3.5 Motion Control With CVXPY

As the limitations of the "CVXGEN" is explained for the mixed integer linear programming technique, it is possible to overcome these restrictions by replacing the optimization stage and the solver with the "CVXPY". The "CVXPY" is defined as a Python-embedded modeling language for convex optimization problems. It is significant to remember that the control algorithm of the quadcopter is created in

the C++ language. The compatibility problem of the two different code language is solved by the benefits of the "ROS Service". As it is mentioned in the previous sections, Robotic Operating System allows us to create Python script node inside the C++ based project. In addition to that, it offers great variety of the solver which are applicable to the different types of the convex problems.

3.5.1 ROS Service

The new implementation of the "CVXPY" is handled by creating an ROS services. The ROS service is one of the communication way between the ROS nodes apart from the classical ROS message type. ROS service has some good advantages which can be listed as[38];

- In the ROS message type of communication, the message is publishing by the node without considering that the message has been received by the other node which is subscriber one. Although, ROS service call is able to wait, detect and collect the response through the communication.

- ROS service call is regulated by the corresponding node where the response of the call also stored. The ROS service can offer one to one communication which can be seen as a difference from the ROS messages.

The concept of the ROS service communication can be visualized as it is given in the following figure;

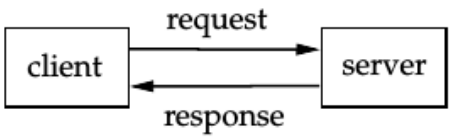

Figure 3.9: ROS Service-Client.
[38]

The client node can be employed to create request which can be seen as data to send the server node. When the server node notices that request from the client node, it begins to execute related defined actions which can be computation, configuration and etc. After that point, the server node sends back the output data to the client which is named as response [38]. So in our implementation two services are created and evaluated. The first service is called as "CvxpyOpt" and the service data structure type is defined in the following code script;

Code Fragment

```
float64[] u_ss
float64[] x_ss
float64[] d
float64[] u_prev
float64[] x_0
---
float64[] u
float64[] x
```

The data u_ss, x_ss, d, u_prev amd the x_0 are used to define steady state control input command, steady state condition of the states, disturbance, previous control input command and initial states respectively and they can be classified as members of the request. After the dash line, control input u and the states x are defined as a response then they addressed to send back when the server completed required actions.

The second service "CvxpyOptInit" is implemented for the initialization sense of the optimization problem. As it is adopted in the "CVXGEN" case, steady state discretizated model of the system should be initialized before then the solver process in order to avoid from the unnecessary computational burden. The data structure of this service can be seen in the following code script.

Code Fragment

```
float64[] Ad
float64[] Bd
float64[] B
float64[] Q_final
float64[] Q_int
float64[] R
float64[] R_delta
float64[] umin
float64[] umax
---
bool success
```

As in the "CvxpyOpt" server structure, before the "–" dash line elements are defined for the request part of the service. Also Boolean type of data is adopted for the response of the service in order to check the initialization is done successfully. After that step, it may be convenient to show how to integrate these two created service in to a "mav_linear_mpc" script. The "CvxpyOpt" service is implemented in the "mav_linear_mpc" script as it is given in the below;

Code Fragment

```
mav_linear_mpc::CvxpyOpt srv;
```

After that definition, it allows us to assigned each of the "CvxpyOpt" service requests which can be as follows;

Code Fragment

```
Eigen::Map<Eigen::Matrix<double, kInputSize,
   1>>(srv.request.u_ss) = target_input;
Eigen::Map<Eigen::Matrix<double, kStateSize,
   1>>(srv.request.x_ss.data()) = opt_queue_[0];
Eigen::Map<Eigen::Matrix<double, kDisturbanceSize,
   1>>(srv.request.d.data()) = estimated_disturbances;
Eigen::Map<Eigen::Matrix<double, kInputSize,
   1>>(srv.request.u_prev.data()) =
   linearized_command_roll_pitch_thrust_;
Eigen::Map<Eigen::Matrix<double, kStateSize,
   1>>(srv.request.x_0.data()) = x_0;
```

After that step, the "CvxpyOpt" service offers a response which is a optimized control input values can be assigned such that;

Code Fragment

```
linearized_command_roll_pitch_thrust_ << srv.response.u[0],
   srv.response.u[1], srv.response.u[2];
```

For that purpose, the client object is created to receive the response from the server. The "private_nh" is knowns as common ROS private node handle object and it allows to integrate created object inside ROS. Then the client object service type is defined as "mav_linear_mpc::CvxpyOpt" also the service name selected as "/cvxpy_solve_opt" which can be seen in the below;

Code Fragment

```
client_ = private_nh_.serviceClient<mav_linear_mpc::CvxpyOpt>
        ("/cvxpy_solve_opt");
```

3.5.2 Opt_Node

In this section, we will briefly explain the created optimization node which is based on the "CVXPY" environment. The given code fragment can be used to describe the initialization of the node and the service server respectively. Also we checked that the initialization is completed without any problem. The "rospy.spin()" is mandatory callback function which executes the main of the code as an active until the node shuts down.

Code Fragment

```
rospy.init_node('cvxpy_optimization_node')
rospy.Service('cvxpy_init_opt', CvxpyOptInit,
   initialize_problem)
rospy.Service('cvxpy_solve_opt', CvxpyOpt, solve)
print "Ready\_to\_solve\_the\_optimization\_problem"
rospy.spin()
```

In the next code fragment it possible to check the initialization of the discrete time model from the created service request can be seen easily. So the "cvxpy_init" is consisted of the given related code fragments. The client node create a request to take the required discrete time model matrix data from the server.

Code Fragment

```
Ad_temp = np.array(req.Ad)
Ad_temp = Ad_temp.reshape(8, 8).transpose()
Ad = sparse.csc_matrix(Ad_temp)

Bd_temp = np.array(req.Bd)
Bd_temp = Bd_temp.reshape(3, 8).transpose()
Bd = sparse.csc_matrix(Bd_temp)

[nx, nu] = Bd.shape

B_temp = np.array(req.B)
B_temp = B_temp.reshape(3, 8).transpose()
B = sparse.csc_matrix(B_temp)

Q_final_temp = np.array(req.Q_final)
Q_final_temp = Q_final_temp.reshape(8, 8).transpose()
Q_final = sparse.csc_matrix(Q_final_temp)

Q_int_temp = np.array(req.Q_int)
Q_int_temp = Q_int_temp.reshape(8, 8).transpose()
Q_int = sparse.csc_matrix(Q_int_temp)

R_temp = np.array(req.R)
R_temp = R_temp.reshape(3, 3).transpose()
R = sparse.csc_matrix(R_temp)

R_delta_temp = np.array(req.R_delta)
R_delta_temp = R_delta_temp.reshape(3, 3).transpose()
R_delta = sparse.csc_matrix(R_delta_temp)

umin = np.array(req.umin)
umax = np.array(req.umax)
```

In the following code fragment, elements of the objective function are defined and the prediction horizon which is ”N” is defined. We will discuss further details about the length of the prediction horizon and the effects of it in the result and discussion part. But for similarity, we can assume the prediction horizon as 20 which is also used by the previous convex solver ”CVXGEN”.

Code Fragment

```
Q = Q_int
R_del = R_delta
Qf=Q_final
delta = Variable((4, N), boolean = True)
N = 20

x = Variable((nx, N+1))
u = Variable((nu, N))
x_0 = Parameter(nx)
u_prev = Parameter(nu)
u_ss = Parameter(nu)
xr = Parameter(nx)
d = Parameter(nu)

objective = 0
constraints = [x[:,0] == x_0]
```

The objective function variable definitions are done as it is given in the above code fragment. Now it is convenient to describe the objective function definition in the below. The constraints are evaluated for the discrete time quadcopter dynamics and for the control input command magnitude as well. It is important to remind that, because of the linear model predictive control algorithm, the optimum control input variable is calculated through the prediction horizon. So practically, it calculates 20 step ahead in the each trajectory point repetitively. Theoretically, model predictive control optimization calculates more accurate and more precise control input command as long as prediction horizon extends. However, it may reveal computational burden which may not be handled with the common used processors.

Code Fragment

```
objective +=quad_form(x_0 - xr, Q) +
   quad_form(u[:,0]-u_prev, R_del)
constraints +=[ x[:,1] == Ad*x_0 + B*u[:,0] + Bd*d ]
constraints +=[ umin <= u[:,0], u[:,0] <= umax ]

for k in range(1, N):
   objective +=quad_form(x[:,k] - xr, Q) + quad_form(
      u[:,k] - u[:,k-1], R_del) #+ quad_form(u2[:,k] -
      u_ss, R)
   constraints +=[ x[:,k+1] == Ad*x[:,k] + B*u[:,k] + Bd*d ]
   constraints +=[ umin <= u[:,k], u[:,k] <= umax ]

objective +=quad_form(x[:,N] - xr, Qf)

prob =Problem(Minimize(objective), constraints)

result =CvxpyOptInitResponse()
result.success =True
return result
```

In the next code fragment, the solve function is defined. Also, the computation time is significant parameter for the quadcopter control and it is counted in order to ensure the compatibility. In addition to that, we decided to use the Gurobi solver which is also used by the global technology companies. The advantage of the "Gurobi" solver can be proposed with the powerful solution performance and the robustness available for the "MILP" problems.

Code Fragment

```
def solve(req):
    global x, u, x_0, u_prev, u_ss, xr, d, prob, delta
    x0 = np.array(req.x_0)
    u0 = np.array(req.u_prev)
    xr_test = np.array(req.x_ss)
    u_ss_test = np.array(req.u_ss)
    t = time.time()
    x_0.value = x0
    u_prev.value = u0
    xr.value = xr_test
    u_ss.value = u_ss_test
    d.value = np.array(req.d)

    prob.solve(solver=GUROBI, warm_start=True, verbose=False)

    elapsed = time.time() - t
    print elapsed
    result = CvxpyOptResponse()
    result.u = u[:, 0].value
    result.x = x[:, 0].value
    return result
```

3.5.3 Obstacle Avoidance with MILP

As we discussed in the previous sections, the ”CVXGEN” does not allow to use Boolean type of the variables in the constraint definitions which is a major requirement to perform mixed integer linear programming techniques in order to have obstacle avoidance on the trajectory planning process.

Mixed Integer Linear Programming (MILP) can be described as very effective and robust mathematical programming technique that contains decision variables which are defined as integer values. Regarding to this decision variables, it is possible to modify trajectory planning optimization through the defined linear constraints[39]. In more practical sense, these binary integer constraints can only be ”0” or ”1” values in order to generate discrete decisions which are used to decide that quadcopter would pass to the right or to the left of the the static object. According to the definition of the MILP, the optimization problem is defined as linear naturally. So, the output of the MILP calculation ensures that the solution is globally optimal[40].

Moreover, in the adopted approach, we decided to define the lower left corner and the upper right corner of the rectangular static obstacles as constraints. The obstacle avoidance could be ensured if the all computed trajectory points of the quadcopter are out of the defined obstacle rectangular areas through the each time step[41].

$$\begin{aligned} x_{min} - x &\geq M\alpha_1; \\ x - x_{max} &\geq M\alpha_2; \\ y_{min} - y &\geq M\alpha_3; \\ y - y_{max} &\geq M\alpha_4; \\ \textstyle\sum_{i=1}^{4} \alpha_k &\leq 3; \end{aligned} \tag{3.10}$$

As it can be seen in the above equations, the rectangular obstacle lower left corner coordinates are defined as x_{min} and x_{max}. The upper right corner is also defined with the y_{min} and y_{max}. It is important to notice that, through the each time steps, at least one of the constraints has to be enforced in order to avoid quadcopter trajectory positions which x and y are not inside the restricted obstacle region. The given constraint enforcing condition is controlled with the last defined constraint. The main objective of the optimum trajectory planning could be assumed as the minimum cost to go principle. Referring to that, quadcopter trajectory plan may be inclined to pass the object very close and also the optimum trajectory may not be calculated during the iterative calculation of the optimization if the quadcopter is located so nearly to the obstacle. So that, one of the convenient ways of tackling that defined problem is to introduce a slack variable which is determined as "M" in the constraint equations.

That "M" slack variable is defined to be sufficiently large where it is employed in the constraints equation. For instance, the binary decision variables which are defined with the $\alpha_{1,2,3,4}$ take "0" or "1" value depends on the which constraint equation will be enforced. So that, when the α takes "0" value, then the variable in the constraint equation is enforced in order to find feasible solution. Contrarily, when the the α takes "1" value, then the constraint equation is relaxed.

In our code implementation, we decided to use "1000" for that M value which you can notice on the code fragment. It may be ensured the feasibility of the optimized trajectory[41]. The obstacle avoidance can be evaluated with the following code fragment substitution in the constraints. Also, the last constraint in the 3.10 equation ensures that at least one constraint equation is enforced.

Code Fragment

```
objective += quad_form(x_0 - xr, Q)
   +quad_form(u2[:,0]-u_prev, R_del)
constraints +=[ x[:,1] == Ad*x_0 + B*u2[:,0] + Bd*d ]
constraints +=[ umin <= u2[:,0], u2[:,0] <= umax ]


constraints +=[ (20)- x[0, k] >= - 10000 * delta2[0, k]]
constraints +=[ x[0, k] - (30) >= - 10000 * delta2[1, k]]
constraints +=[ (20) - x[1, k] >= - 10000 * delta2[2,k]]
constraints +=[ x[1, k] - (30) >= - 10000 * delta2[3, k]]
constraints +=[ delta2[0, k] + delta2[1, k] + delta2[2,
   k] + delta2[3, k] <= 3 ]
```

As it is shown in the below, an example obstacle is defined by the 20 to 30 meter on the x axis and from 20 to 30 on the y axis for the two dimension space. In the result and discussion part we will briefly introduce and comment about the obstacles and the avoidance scenarios as well.

3.6 Computation Specs

The results and simulations are maintained for the computer which has the system specifications and the used software environment version can be given as follows;

- Processor: Intel Core i7-6700HQ CPU 2.60GHz
- Graphics: Intel HD Graphics 530 (Skylake GT2)
- Ram: 16 GB
- Operating System: Ubuntu 16.04 LTS
- Robotic Operating System Kinetic Kane Version May 23rd, 2016

Chapter 4

Results

In Section 4.1, the obtained results from different experiments will be discussed. The effect of prediction horizon length will be discussed through the computation time, the efficiency sense. Also, we will discuss and show the simulations about the capability of the obstacle avoidance algorithm for the multiple obstacle cases and for the different scenarios.

4.1 Experiments

4.1.1 Obstacle Avoidance with One Obstacle

In this simulation, we will test the obstacle avoidance algorithm for the defined target position of the quadcopter with the different prediction horizons in order to illustrate the varied behaviour of the trajectory planning. For that purpose, we designed some proper test case and target points to be ensured about the capability and accuracy of the adopted technique. So it is possible to define the case 1 such that;

- **Obstacle Buffered Location:** The obstacle with buffered zone is colored with red and it is defined between the given distance on the x axis and on the y axis which could be noticed in the each case position tables respectively.
- **Obstacle Actual Location:** The obstacle actual location is colored with black and defined for 1 meter minimized and both for the x and y axis in order to prevent from unfeasible solution and crush.
- **Quadcopter Initial Position:** The quadcopter initial point is always defined as 0 on the x axis and 0 on the y axis for the beginning.
- **Quadcopter Target:** The each target point is defined for the each case which could be seen respectively in the related table.
- **Prediction Horizon Length:** It is decided to test the three different proper prediction horizons which are 20, 30 and 40 for each case.

- **Computation Time:** The computation time will be obtained as minimum, maximum and the average which is an arithmetic mean in order to be ensured about the compatibility of the algorithm inside the control frequencies.

Case 1:

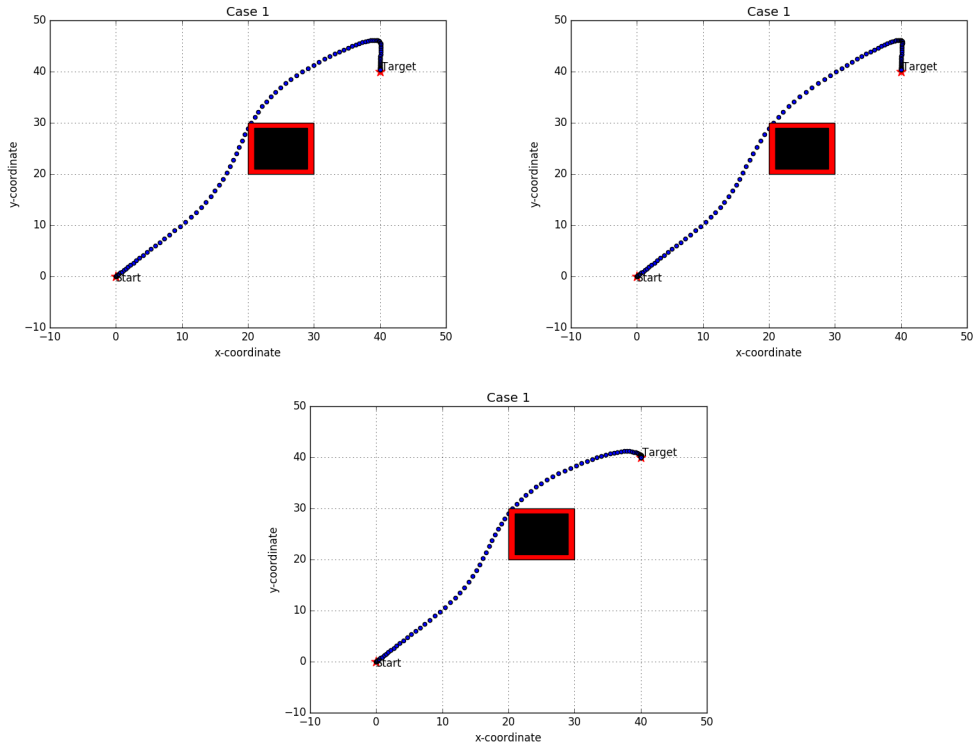

Figure 4.1: Simulation of the different prediction horizon length which are 20 on the upper left figure, 30 for the upper right figure, 40 for the middle down figure for the same target position 40-40.

Position	**x axis**	**y axis**
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Initial	0	0
Target	40	40

Table 4.1: Simulation case 1 position definitions for quadcopter and obstacle with buffer zone.

Computation Time	Minimum	Maximum	Average
Prediction Horizon 20	**0.22 sec**	**0.39 sec**	**0.24 sec**
Prediction Horizon 30	**0.33 sec**	**0.53 sec**	**0.37 sec**
Prediction Horizon 40	**0.45 sec**	**0.76 sec**	**0.51 sec**

Table 4.2: Computation Time Performances case 1.

Case 2:

So it is possible to introduce the case 2 such that;

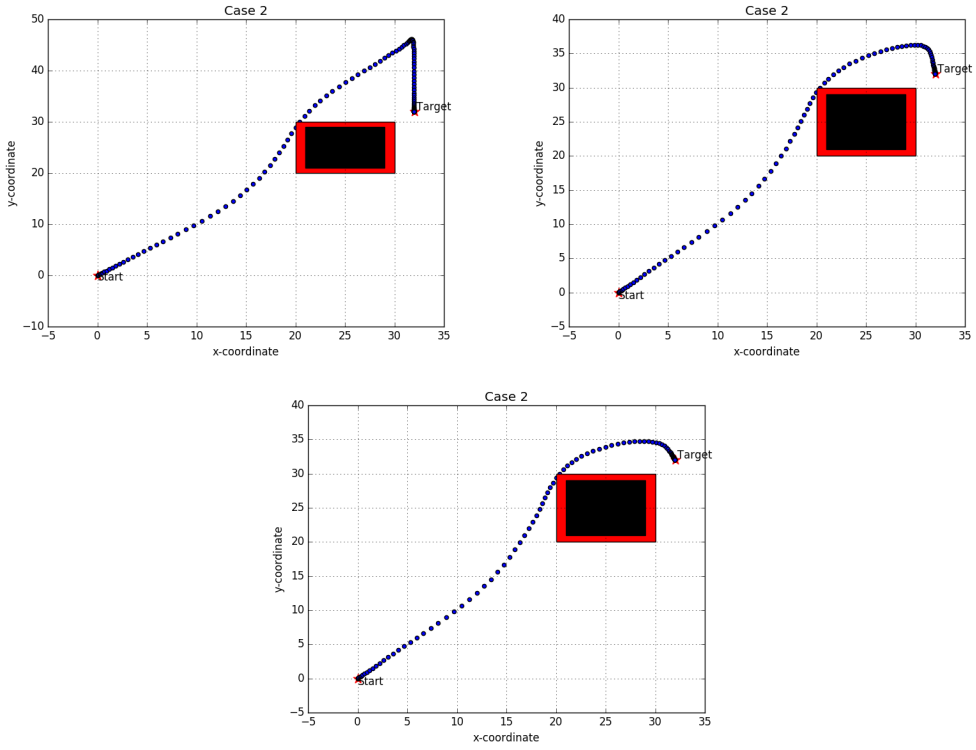

Figure 4.2: Simulation of the different prediction horizon length which are 20 on the upper left figure, 30 for the upper right figure, 40 for the middle down figure for the same target position 32-32.

Position	**x axis**	**y axis**
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Initial	0	0
Target	32	32

Table 4.3: Simulation case 2 position definitions for quadcopter and obstacle with buffer zone.

Computation Time	**Minimum**	**Maximum**	**Average**
Prediction Horizon 20	**0.22 sec**	**0.39 sec**	**0.24 sec**
Prediction Horizon 30	**0.34 sec**	**0.58 sec**	**0.38 sec**
Prediction Horizon 40	**0.46 sec**	**0.77 sec**	**0.54 sec**

Table 4.4: Computation Time Performances case 2.

Case 3:

So it is possible to propose the case 3 such that;

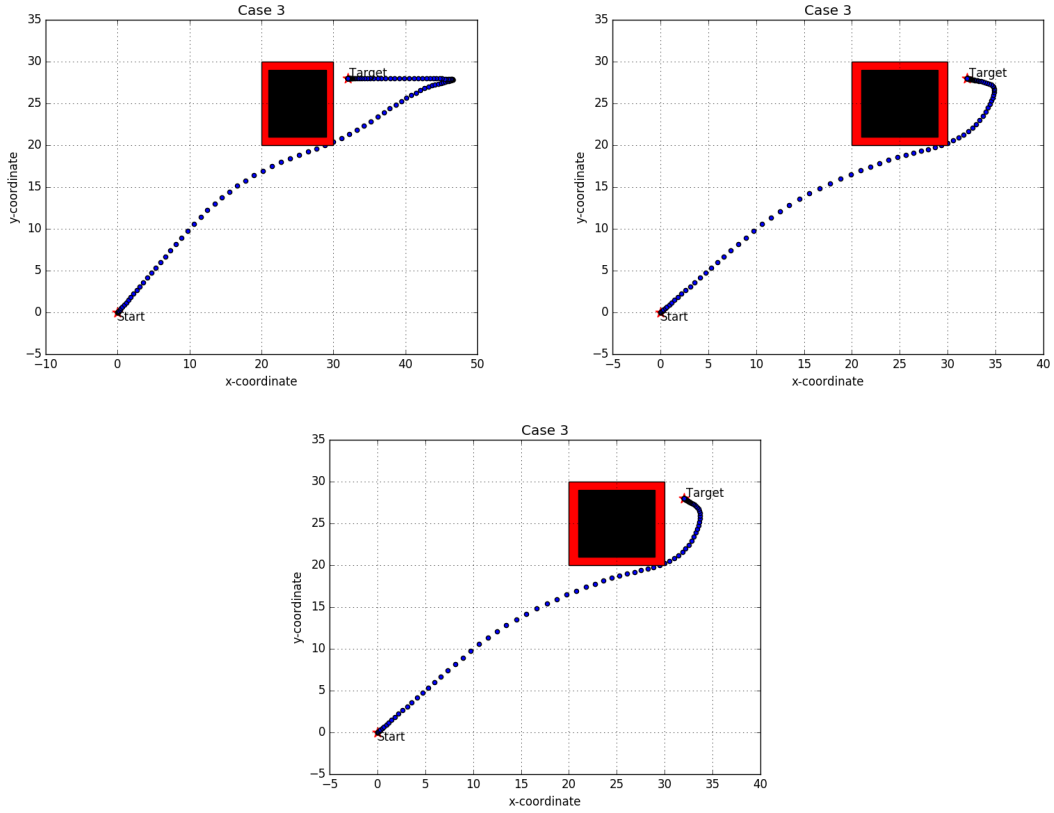

Figure 4.3: Simulation of the different prediction horizon length which are 20 on the upper left figure, 30 for the upper right figure, 40 for the middle down figure for the same target position 32-28.

Position	**x axis**	**y axis**
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Initial	0	0
Target	32	28

Table 4.5: Simulation case 3 position definitions for quadcopter and obstacle with buffer zone.

Computation Time	Minimum	Maximum	Average
Prediction Horizon 20	**0.22 sec**	**0.39 sec**	**0.24 sec**
Prediction Horizon 30	**0.33 sec**	**0.53 sec**	**0.37 sec**
Prediction Horizon 40	**0.47 sec**	**0.81 sec**	**0.53 sec**

Table 4.6: Computation Time Performances case 3.

Case 4:

So it is possible to define the case 4 such that;

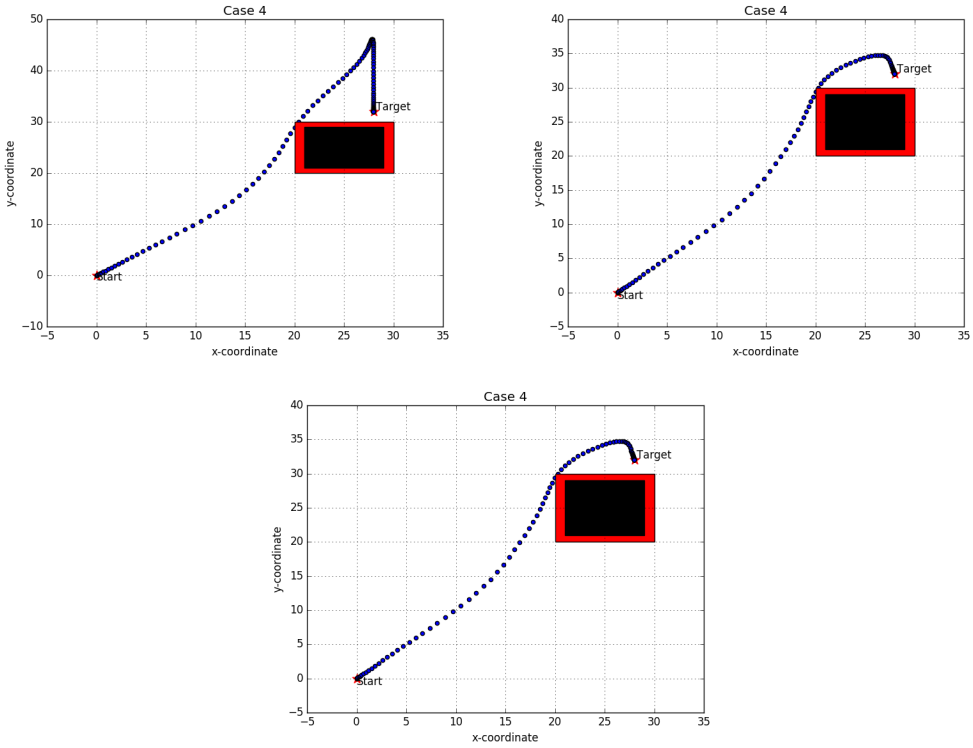

Figure 4.4: Simulation of the different prediction horizon length which are 20 on the upper left figure, 30 for the upper right figure, 40 for the middle down figure for the same target position 28-32.

Position	x axis	y axis
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Initial	0	0
Target	28	32

Table 4.7: Simulation case 4 position definitions for quadcopter and obstacle with buffer zone.

Computation Time	Minimum	Maximum	Average
Prediction Horizon 20	**0.22 sec**	**0.39 sec**	**0.25 sec**
Prediction Horizon 30	**0.34 sec**	**0.54 sec**	**0.38 sec**
Prediction Horizon 40	**0.47 sec**	**0.79 sec**	**0.54 sec**

Table 4.8: Computation Time Performances case 4.

4.1.2 Obstacle Avoidance with Two Obstacle

In this section, we will observe the obstacle avoidance algorithm for the defined target position of the quadcopter with the different prediction horizons in order to notice the change in the behaviour of the trajectory planning. So that, we decided to use similar test scenarios and target points to be ensured about the efficiency and the capability of the derived technique. So it may be convenient to introduce the two obstacle case terms such that;

- **Obstacles Buffered Location:** The each obstacle with buffered zone which is colored with red or blue according to the obstacle and it is defined between the given distance on the x axis and on the y axis which could be noticed in the each case position tables respectively.
- **Obstacles Actual Locations:** The each obstacle actual location is colored with black and defined for 1 meter minimized and both for the x and y axis in order to prevent from unfeasible solution and crush.

- **Quadcopter Initial Position:** The quadcopter initial point is always defined as 0 on the x axis and 0 on the y axis for the beginning.
- **Quadcopter Target:** The each target point is defined for the each case which could be seen respectively in the related table.
- **Prediction Horizon Length:** It is decided to test the three different proper prediction horizons which are 20, 30 and 40 for each case.
- **Computation Time:** The computation time will be obtained as minimum, maximum and the average which is an arithmetic mean in order to be ensured about the compatibility of the algorithm inside the control frequencies.

Case 1:

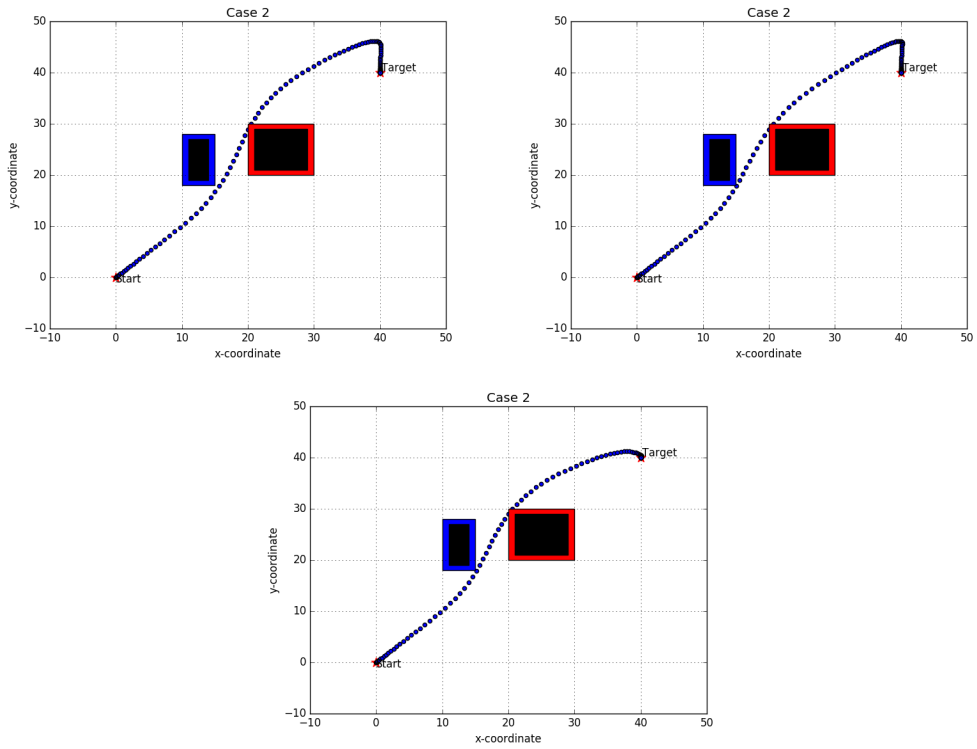

Figure 4.5: Simulation of the different prediction horizon length which are 20 on the upper left figure, 30 for the upper right figure, 40 for the middle down figure for the same target position 40-40.

Position	x axis	y axis
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Obstacle 2 Buffer	10-15	18-28
Obstacle 2 Actual	11-14	19-27
Initial	0	0
Target	40	40

Table 4.9: 2 obstacle simulation case 1 position definitions for quadcopter and obstacle with buffer zone.

Computation Time	Minimum	Maximum	Average
Prediction Horizon 20	**0.31 sec**	**0.52 sec**	**0.36 sec**
Prediction Horizon 30	**0.48 sec**	**0.78 sec**	**0.52 sec**
Prediction Horizon 40	**0.67 sec**	**1.14 sec**	**0.76 sec**

Table 4.10: Computation Time Performances case 1.

Case 2:

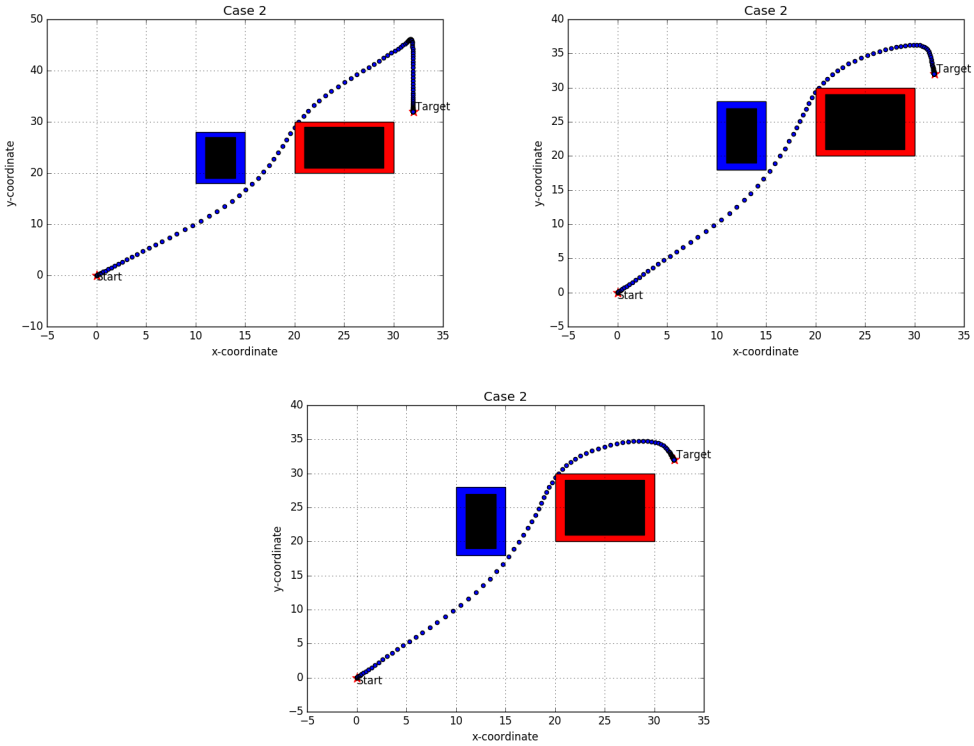

Figure 4.6: Simulation of the different prediction horizon length which are 20 on the upper left figure, 30 for the upper right figure, 40 for the middle down figure for the same target position 32-32.

Position	**x axis**	**y axis**
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Obstacle 2 Buffer	10-15	18-28
Obstacle 2 Actual	11-14	19-27
Initial	0	0
Target	32	32

Table 4.11: 2 obstacle simulation case 2 position definitions for quadcopter and obstacle with buffer zone.

Computation Time	Minimum	Maximum	Average
Prediction Horizon 20	**0.31 sec**	**0.52 sec**	**0.36 sec**
Prediction Horizon 30	**0.48 sec**	**0.78 sec**	**0.52 sec**
Prediction Horizon 40	**0.67 sec**	**1.14 sec**	**0.76 sec**

Table 4.12: Computation Time Performances case 2.

Case 3:

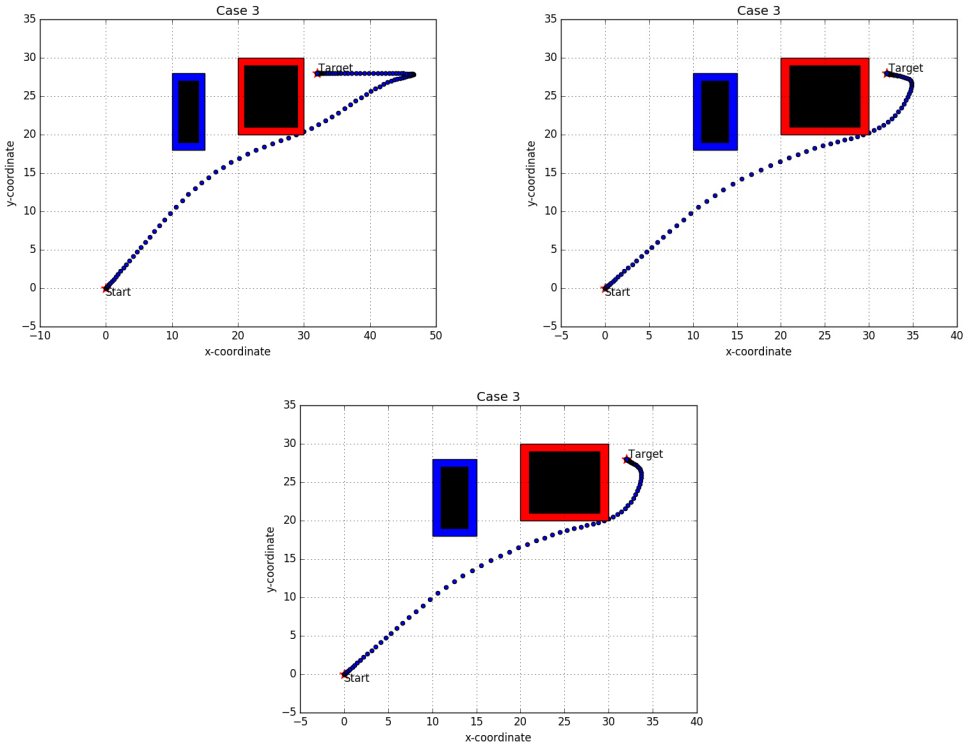

Figure 4.7: Simulation of the different prediction horizon length which are 20 on the upper left figure, 30 for the upper right figure, 40 for the middle down figure for the same target position 32-28.

Position	**x axis**	**y axis**
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Obstacle 2 Buffer	10-15	18-28
Obstacle 2 Actual	11-14	19-27
Initial	0	0
Target	32	28

Table 4.13: 2 obstacle simulation case 3 position definitions for quadcopter and obstacle with buffer zone.

Computation Time	**Minimum**	**Maximum**	**Average**
Prediction Horizon 20	**0.31 sec**	**0.58 sec**	**0.34 sec**
Prediction Horizon 30	**0.48 sec**	**0.80 sec**	**0.54 sec**
Prediction Horizon 40	**0.67 sec**	**1.41 sec**	**0.78 sec**

Table 4.14: Computation Time Performances case 3.

Case 4:

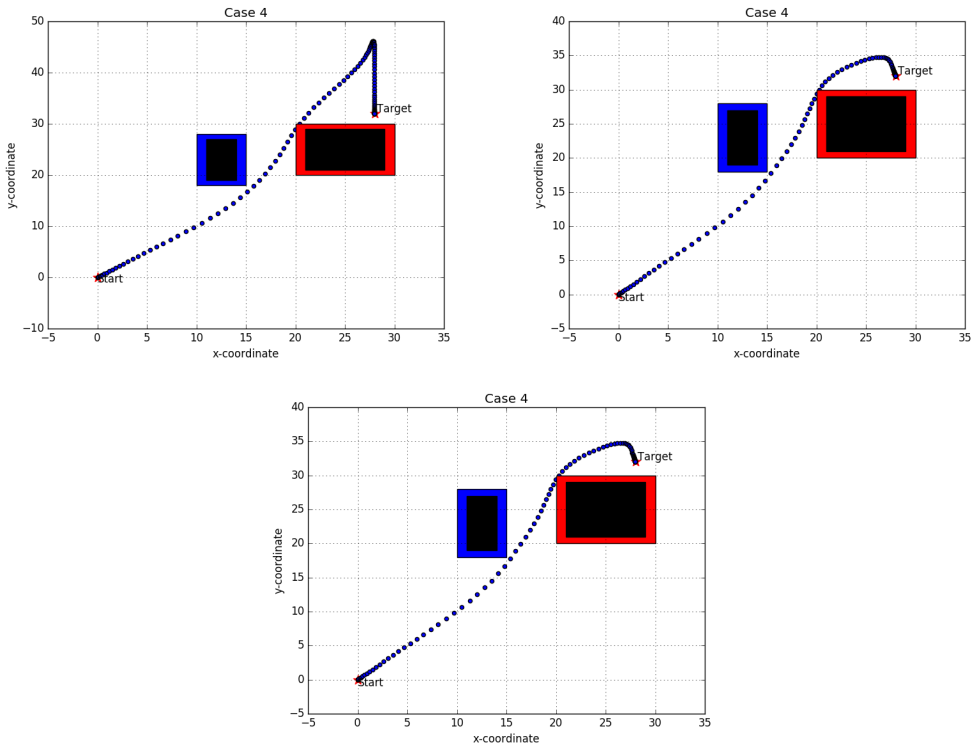

Figure 4.8: Simulation of the different prediction horizon length which are 20 on the upper left figure, 30 for the upper right figure, 40 for the middle down figure for the same target position 28-32.

Position	**x axis**	**y axis**
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Obstacle 2 Buffer	10-15	18-28
Obstacle 2 Actual	11-14	19-27
Initial	0	0
Target	28	32

Table 4.15: 2 obstacle simulation case 4 position definitions for quadcopter and obstacle with buffer zone.

Computation Time	Minimum	Maximum	Average
Prediction Horizon 20	**0.31 sec**	**0.53 sec**	**0.34 sec**
Prediction Horizon 30	**0.48 sec**	**0.80 sec**	**0.53 sec**
Prediction Horizon 40	**0.67 sec**	**1.03 sec**	**0.79 sec**

Table 4.16: Computation Time Performances case 4.

4.1.3 Obstacle Avoidance with Three Obstacle

In the following section, we will present the obstacle avoidance technique for the defined target position of the quadcopter with the predefined and used prediction horizons in order to evaluate the varying character of the trajectory planning points. So, we performed only one test scenario and target point which can be assumed as adequate to show the capability of the obstacle avoidance method through three obstacle collapsed environment. The defined case terms could be presented as follows;

- **Obstacles Buffered Location:** The each obstacle with buffered zone which is colored with red or blue or green according to the obstacle and it is defined between the given distance on the x axis and on the y axis which could be noticed in the each case position tables respectively.
- **Obstacles Actual Locations:** The each obstacle actual location is colored with black and defined for 1 meter minimized and both for the x and y axis in order to prevent from unfeasible solution and crush.
- **Quadcopter Initial Position:** The quadcopter initial point is always defined as 0 on the x axis and 0 on the y axis for the beginning.
- **Quadcopter Target:** The target point is selected carefully to show the obstacle avoidance.
- **Prediction Horizon Length:** It is decided to test only one prediction horizon which is 20 for the case.
- **Computation Time:** The computation time will be obtained as minimum, maximum and the average which is an arithmetic mean in order to be ensured about the compatibility of the algorithm inside the control frequencies.

Case :

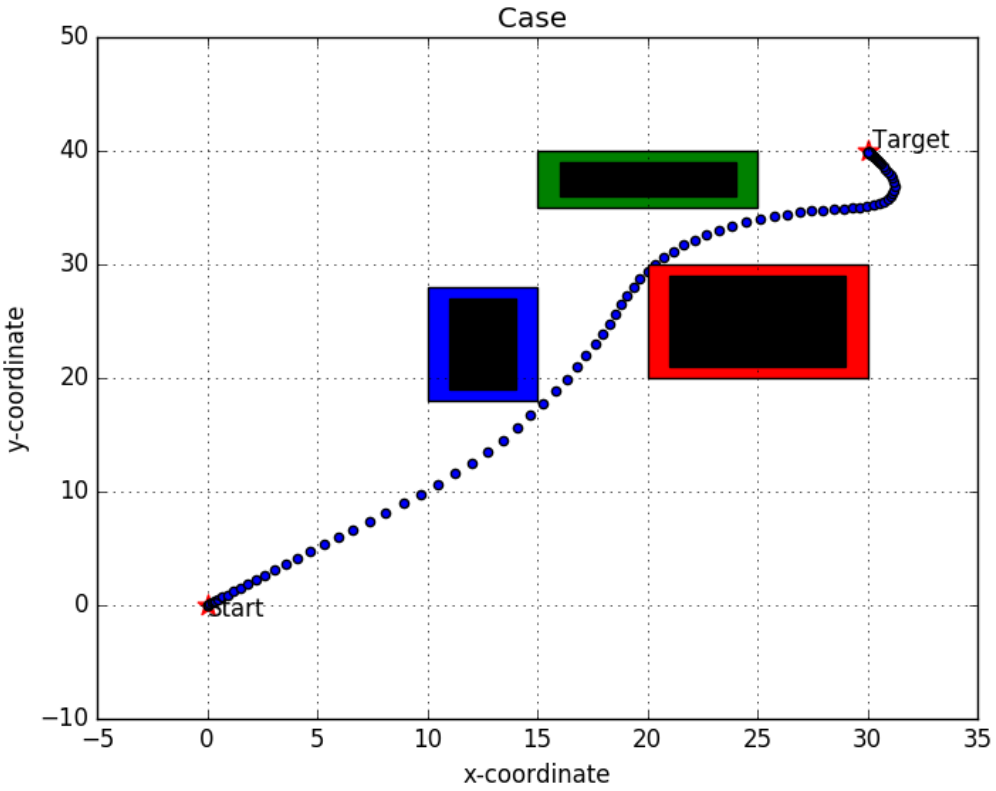

Figure 4.9: Simulation of the prediction horizon length 20 for the target position x:30 y:40.

Position	**x axis**	**y axis**
Obstacle 1 Buffer	20-30	20-30
Obstacle 1 Actual	21-29	21-29
Obstacle 2 Buffer	10-15	18-28
Obstacle 2 Actual	11-14	19-27
Obstacle 3 Buffer	15-25	35-40
Obstacle 3 Actual	16-24	36-39
Initial	0	0
Target	30	40

Table 4.17: 3 obstacle simulation case position definitions for quadcopter and obstacles with buffer zone.

Computation Time	Minimum	Maximum	Average
Prediction Horizon 20	**0.90 sec**	**1.51 sec**	**1.10 sec**

Table 4.18: Computation Time Performance Case.

4.2 Discussion

In this chapter we discuss the given results and the figures in order have better interpretation on the performance of the adopted algorithm which provides motion control and obstacle avoidance capabilities.

4.2.1 One Obstacle Cases

The obstacle avoidance and the optimum path planning capabilities are aimed to test for the one obstacle which is preliminary defined as a constraints for the objective function of the algorithm.

In the Case 1, the obstacle is defined with the buffer zone which can be noticed as red coloured area and the actual obstacle area is illustrated as black colored. The corresponding obstacle is placed from 21 to 29 on x axis and from 21 to 29 on the y axis. The generated optimum paths are varied through the changing length of the prediction horizon in the given Figure 4.1. It is worth to signify that the longer prediction horizon provides better and smoother trajectory points. However, the longer prediction horizon comes with the cost that computation time arises. As it can be seen in the Table 4.2, the different prediction horizon lengths and corresponding computation times are given. From that result table, it may be easy to notice that average computation time when the prediction horizon taken as 40 is almost two times bigger than the prediction horizon 20. So prediction horizon length selection could be adjusted according to the practical application requirements in order to sync the control frequency which is discussed in the internal control loop section.

When it comes into case 2, it is decided to change the target point of the quadcopter into the (32,32) on x axis and y axis to enforce the quadcopter for the more challenging target point. In this case, it was expected to have more aggressive behaviour of the path planning. As it is given in the Figure 4.2, when we adjusted the prediction horizon as 20, we can notice that the behaviour is quite aggressive as we predicted before. This is can be related with the mathematical model of the quadcopter and the physical limitations of the components which are equipped on the quadcopter simulation model. Although, it may be possible state that when the prediction horizon changed for the 40 steps ahead, we can have smoother and more efficient trajectory path ways. It may be important to notice the computation time results which are given in the Table 4.4 that there is no difference in the sense

of computation time for prediction horizon 20 both for the target points which are (40,40) and (32,32). As it is obtained in the case 1, more accurate and proper character of the trajectory can be achieved with the longer prediction horizon which is 40 in this case. On the other hand, the computation time for prediction horizon length 40 could be very demanding for the practical implementations.

In the case 3, it is aimed to propose the changing optimum path planning related to the desired target position for the quadcopter. It is presented in the Figure 4.3, that the target position is changed from 32 to 28 on the y axis and the same position which is 32 for the x axis. Because of the minimum cost to go principle involved in the objective function, the algorithm generates alternative path which is different from the case 1 and case 2 in order to maintain minimum cost for the target position. Additionally, it may be convenient to notice that the behaviour of the generated path is varied according to the selected prediction horizons as well as in the previous cases. The proper and smooth character can be obtained by adopting prediction horizon as 40 but the computation time of that problem complexity could be problem which could not be overcame for the high frequency control system applications.

We decided to keep the same challenging target position strategy where the obstacle remains in the same position like in the case 1, 2 and 3. In the case 4, we only discussed to change the target point in order to notice the change in the cost function that generates efficient and accurate path regarding to the minimum cost to go principle again. In the Figure 4.4, the varied behaviour of the generated path for the different prediction horizons can be shown. For the prediction horizon 20, the generated path points can be identified as not so smooth and efficient for the target position which is given as (28,32) on the x axis and y axis. However, we can have better and more proper character in the prediction horizon 30. Also it may be possible to note that proper and efficient behaviour could be obtained not only for the prediction horizon 40 but also for the prediction horizon 30 in that case. So desired behaviour can be achieved with relatively less computation time cost when the prediction horizon is adjusted into 30 for that scenario.

4.2.2 Two Obstacle Cases

In the two obstacle avoidance capability test scenarios, we decided to keep same target positions which are presented also for the one obstacle avoidance scenarios. The reason behind that selection is to determine how the algorithm changes the generation of the path relying on the new and more challenging conditions.

In the case 1, the new obstacle which can be noticed with the blue colored area with the buffer zone and black colored area for the actual obstacle area. The new obstacle positioned is maintained after carefully consideration which should provide to show two tangential behaviour that proves double obstacle avoidance is obtained through the path plan. As in the previous one obstacle scenarios, the first object placed in the same location where from 20 to 30 on x axis and from 20 to 30 on the y axis. Also the new second obstacle is placed on the x axis from 10 to 15 on x axis and from 18 to 28 on y axis. It is wort to notice that 1 meter buffer zone around the obstacles are placed to increase reliability and avoid from the problems that are related with the sampling time limitation and the physical

limitations of the quadcopter which are integrated in the mathematical model of it. For the given case 1, generated trajectories can be achieved as it is given in the Figure 4.5. It can be easy to state that behaviour of the quadcopter for the given target position is varied related to the prediction horizon settings. Also we can notice that smoother and proper character is achieved in the prediction horizon 40 step ahead for the target position 40 on x and 40 on y axis. In addition, the algorithm may be adjusted to reach goal position in more proper velocities and in more natural way by adopting 40 for the prediction horizon. The two times obstacle avoidance could be interpreted because of the s shape curve of the generated path. However, that obtained satisfactory behaviour of the path comes with more than the two times computation time cost as it can be seen in the Table 4.10 and again it may be important issue for the some specific real applications.

In the case 2, we aimed to force the algorithm for the more challenging target position which is more near to the obstacle to test the capability of the adopted technique. Just like in the one obstacle case 2, we changed the target location into the (32,32) on x axis and y axis. It could be present that the overshoot occurs while the prediction horizon decreases as it is given in the Figure 4.6. As we expected like in the previous cases, using longer prediction horizon increase the smooth and natural way of approaching to reach the desired target position. Not surprisingly, computation time of the given condition increase through the extension of the prediction horizon which can be checked on the Table 4.12.

When it comes into case 3, we tried to test the two obstacle case with the other target position which is taken as same as in the one obstacle case 3. The target position is defined for (32,28) on x axis and y axis. Also we expected to have another trajectory from the case 2 because of the changed cost directly related with the calculated cost on the objective function. So the Figure 4.7 proves that path is changed as like we expected. Additionally, the smoothness also varied because of the chosen prediction horizons. It is clear to see that better character is achieved with the longer prediction horizon which 40 again. However, this trade of between the computation time could be noticed in the Table 4.14 and the expected behaviour can be managed by the designer of the path planning algorithm.

In the last case which is 4, we used exactly same target points like in the one obstacle case 4. So the desired target position placed (28,32) on the the x axis and y axis. The varied behaviour might be seen in the Figure 4.8 where the cornering capabilities are maintained for both two obstacles. Moreover, it is found that algorithm generates similar character which can be satisfactory for the prediction horizon 30 and 40. And according to the Table 4.16, the particular convenience may be offer to use prediction horizon 30 which computation time in 0.53 as average rather than the prediction horizon 40 which computation cost 0.79 on average.

4.2.3 Three Obstacle Case

In this section we will present the test case which 3 obstacles are defined in the path plane. We discussed in the previous that one and two obstacle cases and in that step we decided to add one more obstacle in order to find the computational cost limit which can be reached. Unlikely to the previous test scenarios, we implemented

this test for only one condition that we found after carefully selection that provides to show 3 obstacle avoidance can be achievable with the adopted algorithm. For that reason, we forced the algorithm that we could had three cornering behaviour on the generated path plan. So the red and the blue obstacles are positioned at the same place as like in the previous. The green colored area used to define the third obstacle and also the black colored area is used to present actual obstacle region. The target position is determined as 30 on x axis and 40 on y axis which are evaluated as proper goal points to have desired characteristic on the generated path. When we checked the Figure 4.9, it is possible notice three cornering is maintained for the target for the prediction horizon 20. It is also tested for the longer prediction horizon but as we can see in the Table 4.18, computation time is very demanding even if we set for 20 steps ahead. In the average computation time it reaches to the 1.1 second which may not be convenient to use in the practical application of the quadcopter. However, it may be possible to state that three obstacle avoidance could be enhanced with the derived algorithm.

4.2.4 Error Analysis

In this section, we would like to investigate issues which could be happened during the path planning. For that reason, we decided to take a close look to the generated path by the algorithm that we adopted in this research. In the given Figure 4.10, the generated path points can be identified with the blue point easily. As we can notice, the difference between the blue path points are not equally distributed by the algorithm. This might be explain with the dynamical condition of the drone itself and also with the discretization sampling time preference of the design. The dynamical conditions of the quadcopter are defined with the mathematical model and with the physical limitations which are the components equipped on the model. So it may not be possible to adjust the physical limitations such that the maximum rotor speed. In addition to that, the dimension of the quadcopter is considered inside the mathematical model and it may not be possible to change for some specific conditions.

When it comes into a discretization, that could be adjusted according to the design process of the control system. Moreover, it can be directly related with the sampling time selection of the designer. For instance, in our implementation we followed and designed the external control loop where the linear model predictive controller performs that must be compatible with the internal control loop frequency. Otherwise, there might be some uncertainties and uncontrollable situations which may drives the system into a instability so quickly.

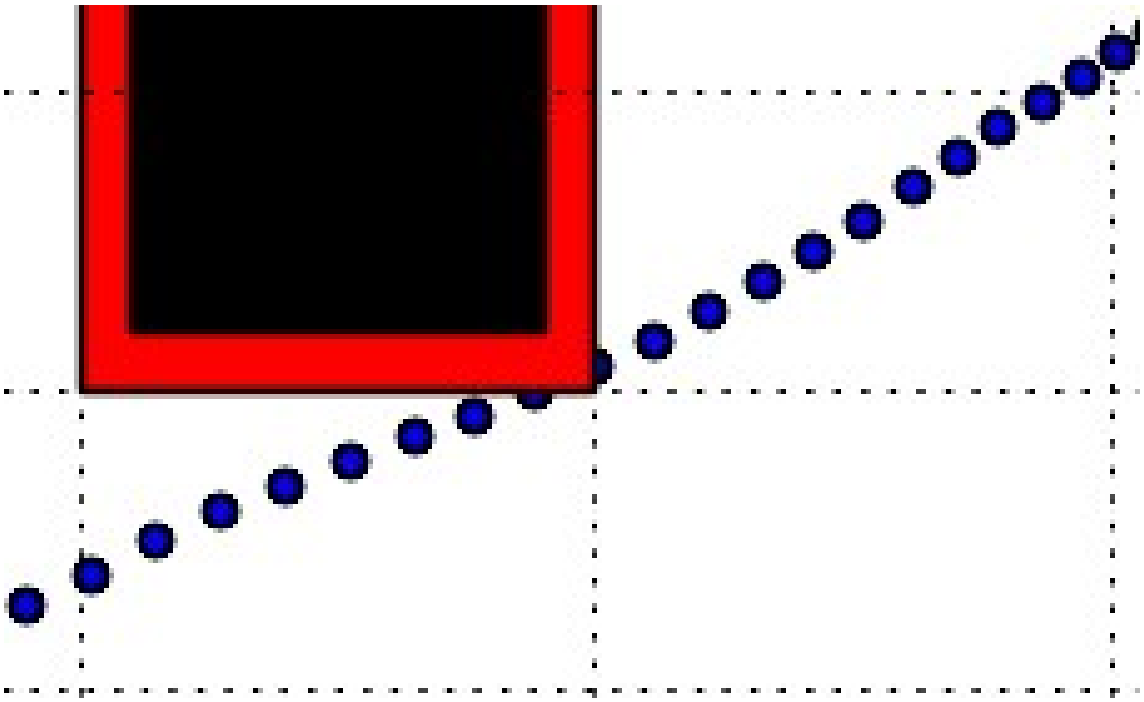

Figure 4.10: Error Analysis.

In the given Figure 4.10, the explained possible issues can be tackled by introducing the buffer zone which is defined that surrounds the obstacle with the red colored area. That buffer area is created to be 1 meter away from the actual position of the obstacle both for each corner of it. As we can notice on the Figure 4.10, some of the generated path points or the distance between the two blue path points might be inside that red colored area, especially, when the quadcopter reaches to pass the corner of the obstacle. So that, we designed an buffer zone surround the obstacle in order to avoid the quadcopter from the issues which are explained at the above.

4.3 Simulation Environment & Simulation Images

In the following section, we will explain simulation details of the developed algorithm which is based on the mixed integer linear programming technique for the obstacle avoidance and path planning as we mentioned before. For that purpose, Gazebo simulation environment is adopted which could be fitted with the application requirements. Gazebo simulation environment can be defined as a dynamic simulator which is capable to allow to simulate indoor and outdoor conditions through great variety of different robot models[42]. Additionally, it can perform and simulate accurate physical dynamics response of the model by considering the accurate sensor dynamics as well. Since Gazebo is a open source developed project provided by the researchers and users, it could always be convenient to find the proper solutions for the specific problems and issues thanks to that contribution of the community. For that reason, the use of gazebo arises particularly among the robotic system researchers and developers[43].

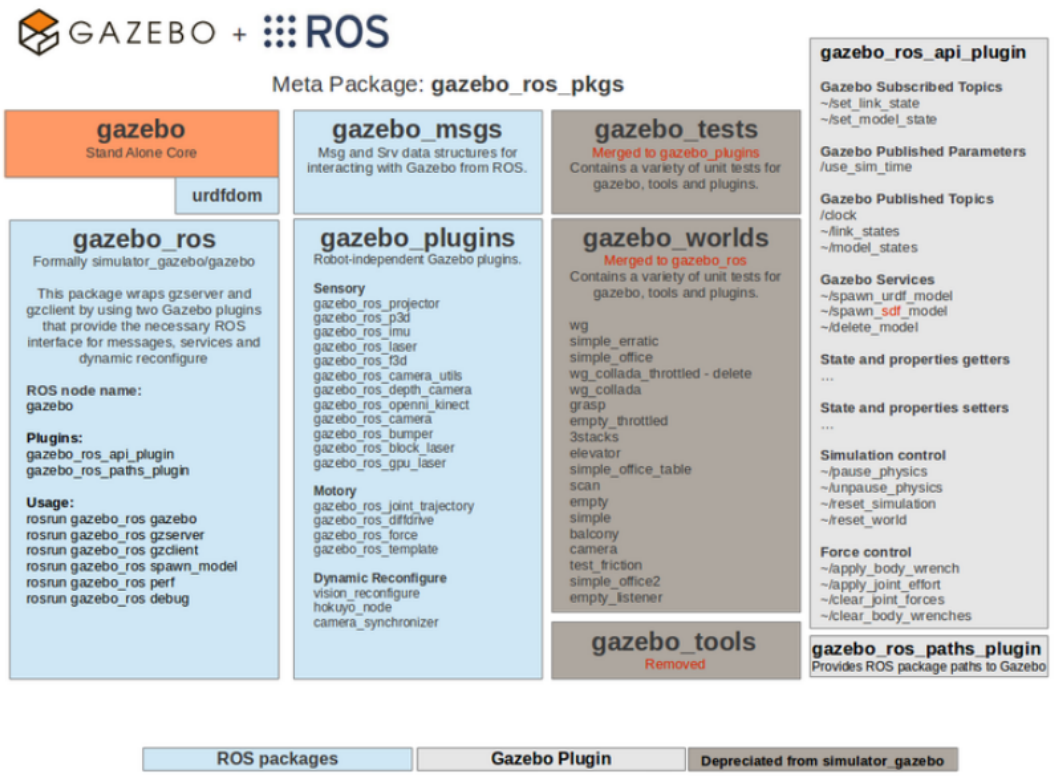

Figure 4.11: Gazebo Simulation with ROS[44]

In the given Figure 4.11, the Gazebo simulation integration to the ROS could be illustrated by the given interface diagram of the package which provides communication between the ROS and the Gazebo. Thanks to the that package, it may be possible control of the interface which simulates the robot in the Gazebo simulation environment by adopting ROS messages and services[44]. In the corresponding package, it may offers great freedom to modify and change conditions due to specific requirements of the simulation. Also, it may performed high quality graphics output thanks to the robust and efficient performance of the Gazebo engines.

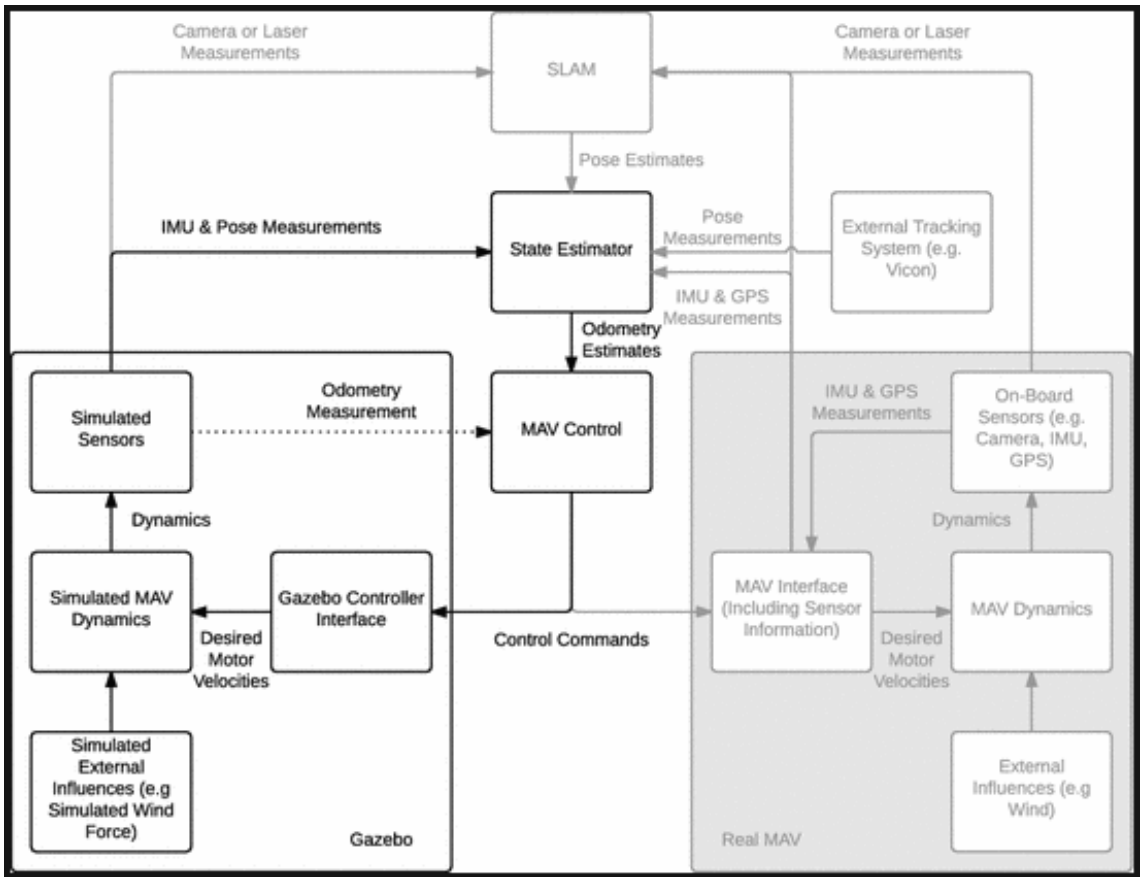

Figure 4.12: RotorS Simulation with ROS.[45]

In more detail, we followed a specific type of the gazebo simulator which is compatible with the ROS. Corresponding simulator which Rotors is developed by the [45] allow us to use well modelled and defined varied multicopters. As it is mentioned before, it is decided to implement path planning and obstacle avoidance capability for the quadcopter model of the UAV. Also, the followed simulator provides accurate simulation outputs of the simulated sensor which are IMU, stereo camera, odometry sensor and they can be configured according to the specific application. In addition to that, given simulator provides a direct acces to the each sensor data that may be convenient in the sense of debugging and accessibility[45]. As it is given in the above Figure 4.12, it is possible to show the RotorS simulator overview. According to the [45], each of the given simulator structure components are developed in a way that each of the components can be operated on the real platform without any changes.

4.3.1 Simulation Images

According to the described Gazebo simulation environment and specifications, some of the simulation image snapshots of the improved obstacle avoidance capability can be given as below;

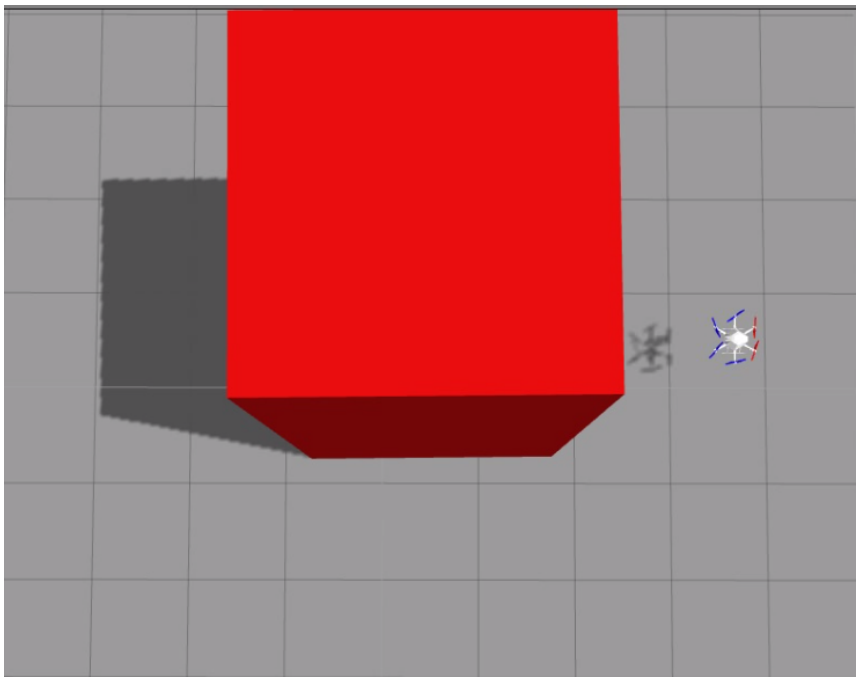

Figure 4.13: Simulation Snapshot 1.

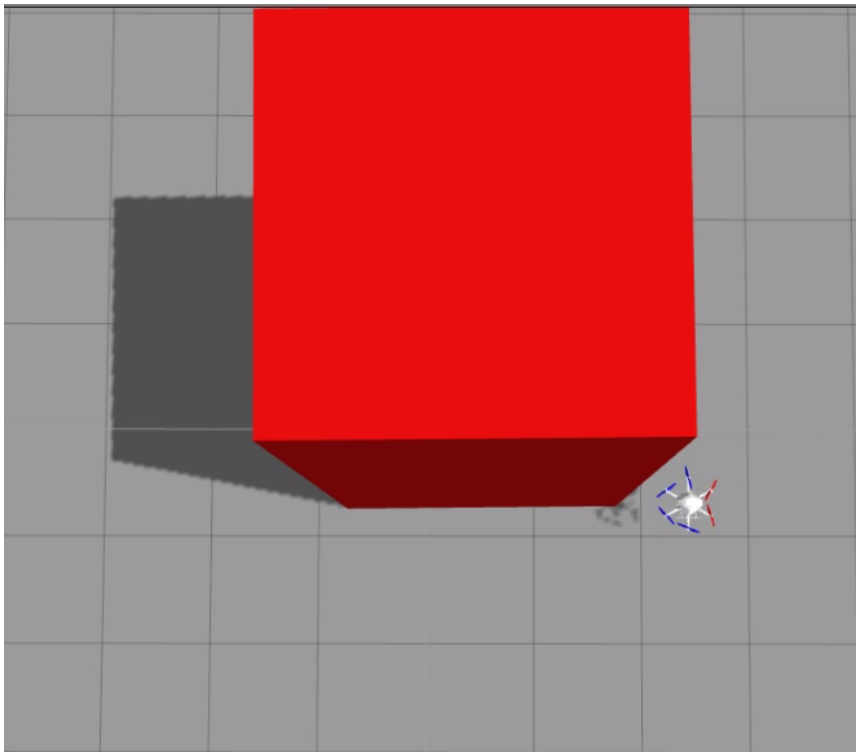

Figure 4.14: Simulation Snapshot 2.

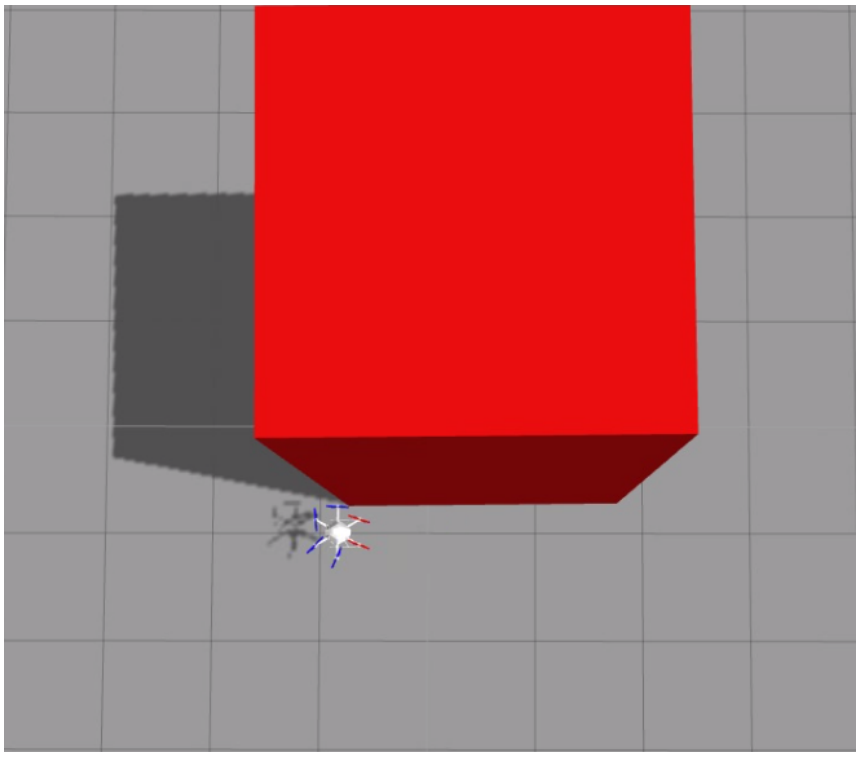

Figure 4.15: Simulation Snapshot 3.

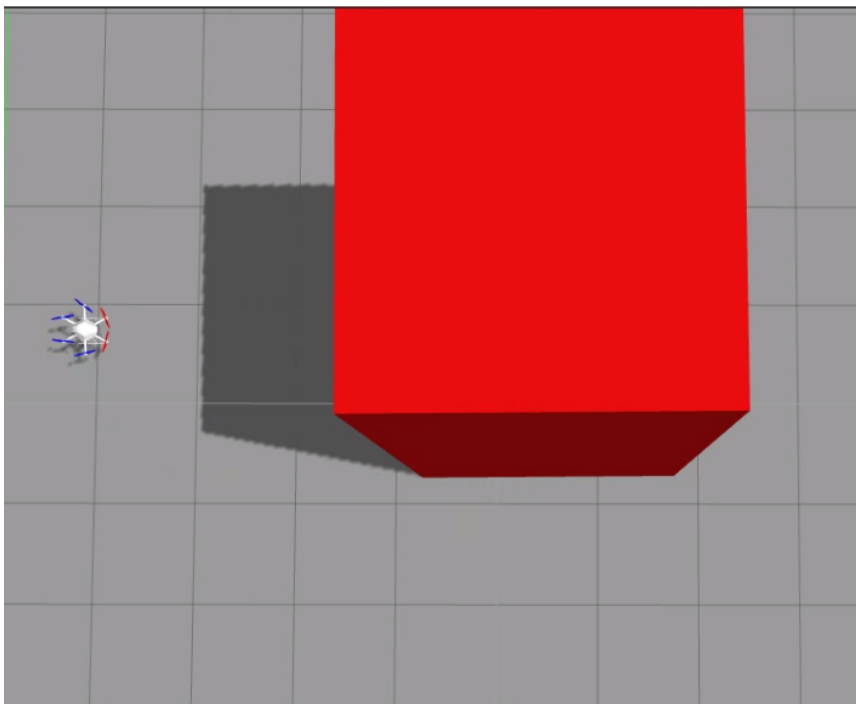

Figure 4.16: Simulation Snapshot 4.

Chapter 5

Conclusion and Future Work

The UAV technology, which was driven for the advance in the military technology at the beginning era, have been developing and expanding through the different potential application areas due to the fact that it may provide a solution to the challenging engineering problems in an alternative way. Based on this motivation, the concepts of the UAV control techniques have been investigated and the obstacle avoidance motion control capability of the proposed quadcopter has been developed, discussed and tested through the varying scenarios in this project. The project based on varying background information which are presented and discussed through the thesis report. Robotic operating system is one of the concept which is involved in the project. It was required to understand the relations,hierarchy and the capability of that environment in order to manage and develop according to the our goal.

On the other hand, some mathematical modelling concepts are covered during this project. It is important to understand the advantages and drawbacks of the adopted mathematical modelling approaches. The concept of the linear model predictive control is discussed and evaluated in detailed and the conformity of the control approach was also considered for our goal. As we referenced before, we followed the project which was developed by the [18], [32] at the "ETH Zurich" and we developed that system for the obstacle avoidance capability in the robotic lab at the Politecnico Di Torino. It was clear that there were specific performance limitations which are directly related with the determined performance capability of the quadcopter model and with the convex optimization solver in the beginning.

Furthermore, we were capable to change the optimization step by creating new Robotic Operating System(ROS) node. Thanks to the developed node, we applied mixed integer linear programming technique in order to make obstacle avoidance capability through the path planning of the quadcopter. Once the system was implemented, it was possible to test the design and obtain the results through simulations that have been reported in this thesis in Chapter 5. All the test scenarios are measured in the sense of computation time to have better interpretation about the conformity of the selected approach. As it is discussed in the previous, one of the major problem was the computation time of the optimization process. Since the control frequency is limited for the optimization calculations for the real applications.

As the time is limited for the thesis project, it might not be convenience to determine very specific details about the subsystems of the whole project. Some other limitations might be occurred because of the kinodynamic constraints which are addressed to the physical limits of the components equipped on the quadcopter. There is no doubt that the control design and the components of the vehicle should be selected properly according to the defined performance requirements of the desired applications.

In the future research, the implemented Off-board Model Predictive Control approach can be realized on the Cloud based systems. Consequently, we will have more computational resources and improved performance with the concept of Cloud Computing. Additionally, we assume that the performance of the obstacle avoidance capability may be improved with the better processor performance which would be found in the future market, what's more, it would be possible to adopted better optimization solver tools which would be offered by the developers and researchers and also there might be more sophisticated sensor communication and battery improvements that provides enlarging in the application area for the UAV in the future.

Acknowledgement

I would like to thank my thesis advisor Prof. Alessandro Rizzo of The Department of Electronics And Telecomunications at the Politecnico Di Torino for lead me to work on this project. I would like to special thank to my advisor Dr. Stefano Primatesta for his invaluable feedback during the development stage of my thesis. He was always been very helpful and perfect mentor that he provided me a proper pathway with his experiences and with his knowledge which was required to complete this thesis successfully.

I am also grateful to Berkay Çalışır, Çagrı Karakuş, Serkut Şimşek and Berk Bayam for their priceless friendship and supporting me through my master education duration since I met with them.

Huge thanks to Burcu Cesur, for her infinite encouragement and unfailing support throughout my years in university and motivate me during my pleased and hard times. I am very lucky to have you in my life.

Finally, I would like to thank my parents for their endless, limitless and priceless support in every moment of my life. Without them, any of my achievements would not have been possible to have.

Kamil Umur Güzel
July, 2019
Torino

Bibliography

[1] John Buckley. *Air power in the age of total war*. Routledge, 2006.

[2] Anthony Finn and Steve Scheding. Developments and challenges for autonomous unmanned vehicles. *Intelligent Systems Reference Library*, 3:9–33, 2010.

[3] John William Ransom Taylor and Kenneth Munson. *Jane's pocket book of remotely piloted vehicles: robot aircraft today*. Collier Books, 1977.

[4] Lee Pearson. Developing the flying bomb. *Naval Aviation in World War I*, pages 70–73, 1969.

[5] David Donald. Encyclopedia of world aircraft (etobicoke, ontario, 1997.

[6] P Werrell Kenneth. The evolution of the cruise missile. *Washington DC: Air University*, page 29, 1985.

[7] Alan Bramson and Neville Hamilton Birch. *The Tiger Moth Story*. Airlife, 1982.

[8] Ben Zimmer. The flight of âĂŸdroneâĂŹfrom bees to planes. *The wall street journal*, 26, 2013.

[9] Laurence R Newcome. *Unmanned aviation: a brief history of unmanned aerial vehicles*. American Institute of Aeronautics and Astronautics, 2004.

[10] Ulrike Esther Franke. Civilian drones: Fixing an image problem? *ISN Blog. International Relations and Security Network. Retrieved*, 5, 2015.

[11] Janosch Nikolic, Michael Burri, Joern Rehder, Stefan Leutenegger, Christoph Huerzeler, and Roland Siegwart. A uav system for inspection of industrial facilities. In *2013 IEEE Aerospace Conference*, pages 1–8. IEEE, 2013.

[12] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *49th IEEE conference on decision and control (CDC)*, pages 5420–5425. IEEE, 2010.

[13] Agus Budiyono. Advances in unmanned aerial vehicles technologies. In *International symposium on intelligent unmanned system*, pages 1–13, 2008.

[14] Michael Blösch, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Vision based mav navigation in unknown and unstructured environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 21–28. IEEE, 2010.

[15] Samir Bouabdallah, Andre Noth, and Roland Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2451–2456. IEEE, 2004.

[16] J Guldner and VI Utkin. The chattering problem in sliding mode systems. In *Fourteenth Intenational Symposium of Mathematical Theory of Networks and*

systems, MTNS2000, 2000.

[17] Samir Bouabdallah and Roland Siegwart. Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2247–2252. IEEE, 2005.

[18] Mina Kamel, Michael Burri, and Roland Siegwart. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017.

[19] Alberto Bemporad, Carlo A Pascucci, and Claudio Rocchi. Hierarchical and hybrid model predictive control of quadcopter air vehicles. *IFAC Proceedings Volumes*, 42(17):14–19, 2009.

[20] Paul Pounds, Robert Mahony, Peter Hynes, and Jonathan M Roberts. Design of a four-rotor aerial robot. In *Proceedings of the 2002 Australasian Conference on Robotics and Automation (ACRA 2002)*, pages 145–150. Australian Robotics & Automation Association, 2002.

[21] S Joe Qin and Thomas A Badgwell. An overview of industrial model predictive control technology. In *AIChE Symposium Series*, volume 93, pages 232–256. New York, NY: American Institute of Chemical Engineers, 1971-c2002., 1997.

[22] Eduardo F Camacho, Carlos Bordons, and M Johnson. Model predictive control. advanced textbooks in control and signal processing, 1999.

[23] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: theory and practiceâĂŤa survey. *Automatica*, 25(3):335–348, 1989.

[24] Kenneth R Muske and James B Rawlings. Model predictive control with linear models. *AIChE Journal*, 39(2):262–287, 1993.

[25] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[26] Graham Goodwin, María M Seron, and José A De Doná. *Constrained control and estimation: an optimisation approach*. Springer Science & Business Media, 2006.

[27] H.D. Cheng, Xiaopeng Cai, Xiaowei Chen, Liming Hu, and Xueling Lou. Computer-aided detection and classification of microcalcifications in mammograms: a survey. 36:2967–2991, 12 2003.

[28] SS a Keerthi and Elmer G Gilbert. Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations. *Journal of optimization theory and applications*, 57(2):265–293, 1988.

[29] Zoran Benić, Petar Piljek, and Denis Kotarski. Mathematical modelling of unmanned aerial vehicles with four rotors. *Interdisciplinary Description of Complex Systems: INDECS*, 14(1):88–100, 2016.

[30] M Belkheiri, A Rabhi, A El Hajjaji, and C Pegard. Different linearization control techniques for a quadrotor system. In *CCCA12*, pages 1–6. IEEE, 2012.

[31] Mark L Darby and Michael Nikolaou. Mpc: Current practice and challenges. *Control Engineering Practice*, 20(4):328–342, 2012.

[32] Mina Kamel, Thomas Stastny, Kostas Alexis, and Roland Siegwart. Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system. In *Robot Operating System (ROS)*, pages 3–39. Springer,

2017.

[33] Tiago P Nascimento, Carlos Eduardo Trabuco Dórea, and Luiz Marcos G Gonçalves. Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach. *International Journal of Advanced Robotic Systems*, 15(1):1729881418760461, 2018.

[34] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[35] Christoph Trabert, Andreas Ulbig, and Göran Andersson. Model predictive frequency control employing stability constraints. In *2015 American Control Conference (ACC)*, pages 5678–5685. IEEE, 2015.

[36] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[37] Jacob Mattingley and Stephen Boyd. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[38] Jason M O'Kane. A gentle introduction to ros. 2014.

[39] Arthur Richards and Jonathan P How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 3, pages 1936–1941. IEEE, 2002.

[40] Christodoulos A Floudas. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[41] Alexander Papen, Ray Vandenhoeck, Jan Bolting, and François Defay. Collision-free rendezvous maneuvers for formations of unmanned aerial vehicles. *IFAC-PapersOnLine*, 50(1):282–289, 2017.

[42] Gazebo gazebo. `http://gazebosim.org/tutorials?tut=guided_b1&cat=`.

[43] Luigi Mazzara. *Risk-aware path planning and replanning algorithm for UAVs*. PhD thesis, Politecnico di Torino, 2018.

[44] Gazebo gazebo ros simulation. `http://gazebosim.org/tutorials?tut=ros_overview&cat=connect_ros`.

[45] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. RotorsâĂŤa modular gazebo mav simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer, 2016.