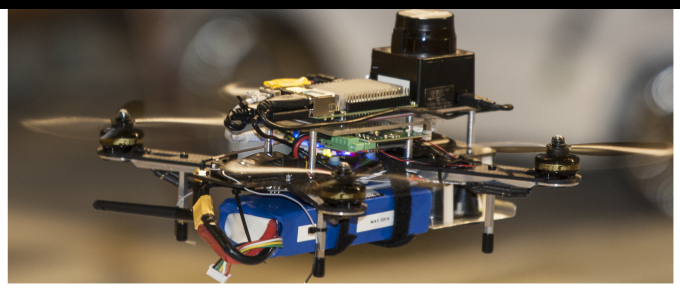
CONFIDENTIAL

A General Purpose Control Design For Vision Based Autonomous Quadrotor Navigation

Manuel Rucci

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

MSCCONFIDENTIAL

A General Purpose Control Design For Vision Based Autonomous Quadrotor Navigation

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Manuel Rucci

October 28, 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work described in this thesis was carried out in cooperation with the Computer Vision Aerial Robotic group of the Technical University of Madrid. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

A General Purpose Control Design For Vision Based Autonomous Quadrotor Navigation

by

Manuel Rucci

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: October 28, 2017

Supervisor(s):

prof.dr.ir. Pascual Campoy

Reader(s):

prof.dr.ir. Javier Alonso Mora

dr.ir. Joris Sijs

ir. Christophe De Wagter

Abstract

Quadrotors are unmanned aerial vehicles (UAVs) which are controlled by changing the angular speed of the four rotors. In the recent years, they have received a large attention from the research community thanks to their vertical take-off and landing capability, small dimension, payload, flight endurance and low price which it has increased the interest in making them completely autonomous. In this thesis two different problems related to the autonomous quadrotor navigation field have been studied. The first one is how it is possible to drive the quadrotor from its current pose (position and orientation) to the desired one assuming that the quadrotor's full states measurements (position, orientation,velocity and acceleration) are available. The second problem deals with the use of visual informations either acquired by the front or bottom camera to place the quadrotor at a desired distance from a chosen static or moving object keeping the latter centered in the acquired camera image without knowing neither the quadrotor's pose nor the object one. To solve the first problem a navigation controller framework able to communicate with both parrot AR. Drone 2.0 and Pixhawk autopilot has been designed. A cascade control design made up by seven 2DOF PID controllers divided into five different modules (horizontal position controller, vertical position controller, horizontal speed controller, vertical speed controller, yaw controller) has been developed to ensure that the navigation controller framework is able to simultaneously track the desired yaw angle ($\psi^\star$) together with either the desired velocities ($\dot{x}^\star, \dot{y}^\star, \dot{z}^\star$) or the positions ($x^\star, y^\star, z^\star$). To solve the second problem a vision based planner made up by three different modules (perception, image state estimator, image based visual servo) has been developed. Planar ArUco markers have been used to avoid the need of designing different specific objects' detectors. Their corners have been extracted by the ArUco detector located inside the perception module. Inside the image state estimator module a Kalman filter with velocity constant model has been designed for both estimating the corners when a detection is suddenly not available and providing to the image based visual servo module feedbacks data (estimated corners) at a specific chosen frequency. An image based visual servo algorithm has been implemented to compute the desired quadrotor's translational velocities that the navigation controller framework has to track to minimize the error between the estimated and the desired corners where the latter are computed as a function of the desired distance between the quadrotor and the visual marker. To validate the navigation controller framework and the vision based planner a mis-

sion has been chosen in which firstly the quadrotor autonomously reaches a first predefined desired pose, secondly it approaches up to a desired distance a static visual marker using the front camera image data, thirdly it reaches a second predefined desired pose and finally it autonomously lands using the bottom camera image information either on a static or moving visual marker. The mission has been tested in real flight with Parrot AR. Drone 2.0 and in simulation using Gazebo simulator in combination with PX4 Software-In-The-Loop. Both the navigation controller framework and the vision based planner have been developed using ROS as a software framework to guarantee system communication and all the algorithms have been coded using C++ as a programming language.

Table of Contents

List of Figures

List of Tables

Preface

This thesis is the result of an amazing experience that I have the pleasure to live in the city of Madrid. In this beautiful city I met brilliant people coming from different countries and having different academic backgrounds. I came to Madrid with the idea of doing an internship of three months and I remained there for one year and a little bit more. The choice to remain and start my thesis in the Computer Vision Aerial Robotics from the Technical University of Madrid was not difficult to make. I was surrounded by brilliant, genuine and expert PhD students that every day were answering about my stupid questions spending some time in teaching something new. If I think on how this thesis has been developed the answer is really simple. It is basically the result of a series of problems that I had to face to make the algorithm works. In the beginning I was focusing my attention in designing a vision-based planner algorithm to solve object following and autonomous landing missions. In doing this the implementation was a mess and I start facing big problems in using my algorithms combined with different algorithms of the group. Thus, I start paying more attention on how to develop a code that was in tune with the idea of the Aerostack. The latter is an open source framework developed by the group allowing to fully autonomous control different type of UAVs. Learning how the Aerostack works allowed me to modify my code and make it more modular and above all readable. Furthermore, I realize the power of the Aerostack mainly when I was able to combine my module in a simple way with the other one already available. Engineering is a group work and I think this is the biggest lesson that I learn this year. After having integrated the vision based planner algorithm inside the Aerostack I start working with the Aerostack controllers. Discussing with the people of the group we realize that it was required to modify the design of the Aerotack controller architecture especially because it was not easy to tune the controllers and replace them with new ones. Thus, together with the other people of the group we start thinking how to modify the Aerostack control architecture to ensure autonomous UAV navigation. Having designed everything according to Aersotack conventions allowed me to combine the new designed navigation controller architecture with the vision based planner and generate different type of mission among which object following and autonomous landing ones. In this thesis I will present both the navigation controller architecture and the vision based planner.

Acknowledgements

There is a huge amount of people that I would like to thank for having spend part of their own time in teaching me things that were banal for them. I have really appreciate that and from academic point of view I am really glad to have had the opportunity to work with so brilliant and genuine people.

Firstly, I would like to say thank you to my professor supervisor Pascual Campoy for giving me the opportunity to come to Madrid and join to the Computer Vision Aerial Robotics group. Secondly, I would like to say thank you to Hriday Bavle an Indian PhD student speaking Spanish far better than me. He teaches me everything about both ROS and the Aerostack software framework investing a great deal of time answering to my stupid questions. Thirdly, I would like to say thank you to the PhD student Alejandro Rodriguez Ramos. He teaches me everything I know about Gazebo simulation environment. Fourthly, I would like to say thank to Jose Luis Sanchez Lopez from which I learn the importance of having code properly organize and structured as it is in the Aerostack.

It is time to say thank you to the PhD Carlos Sampedro Pérez from which I learn how to program properly in $C++$ reading lots of it is code. He also helps me a lot for every problem related to computer vision. Other two PhD students that have really supported me and helped me during this year are Adrián Carrio and Ramón Suárez Fernández . The first one help me a lot in designing the vision based planner and the second in designing the controller architecture. Without them I will never have developed a thesis like this one. They listen to all my problems and discuss with me possible solutions providing me a great deal of suggestions. I really have to say thank you to both of them for their help.

If I grow up as a person this year it is because I share this experience in Madrid with special people. If I think to this people thousand of beautiful memories of this experience in Madrid rise and each of them is able to make me smile. I am speaking about Lorenzo Bracco, Maddalena Giglia, Danila Pilastro, Pedro Ojea, Anna Mingozzi, Irene Bolanos, Maria Stella Cipriani, Alessandro Moroldo, Sergio Vivarelli, Alex Orsenigo and Maddalena Parente.

Delft, University of Technology — Manuel Rucci
October 28, 2017

"Why do you love Engineering? Because It is just a big problem that cannot be solved in one single shot by only one person. It has to be decomposed in really small subproblems that are so many that a single person will take all his/her life to solve them. Different people have to cooperate and trust each other to solve different subproblems and get a result that a single person will never be able to reach in the same period of time."

Chapter 1

Introduction

1-1 Motivation

The works developed in this literature has been carried out at the Center for Automation and Robotics with the Computer Vision Aerial Robotics [1] group from the Technical University of Madrid. The group has a background in Computer Vision mainly oriented to aerial robotics. The main focus of this group is to develop an aerial robotic opens source platform called Aerostack[2] [1] [2] aiming at simplifying the design of complex aerial robotic missions such as inspection of indoors environments, transportation of objects either in indoor or outdoor environments, search and rescue and exploration of unknowns and dangerous environments. The main objective of this group is to develop algorithms aiming at increasing the level of autonomy of UAVs. Among the different types of UAVs the group is mainly oriented in studying a specific type of UAV called quadrotor. To be able to achieve autonomous quadrotor navigation three different area of research have to cooperate with each other.

1. **State Estimation** area of research.

   This area of research focuses on the estimation of the position (x, y, z), the velocity $(\dot{x}, \dot{y}, \dot{z})$ and the orientation (ϕ, θ, ψ) of the quadrotor with respect to a fixed world coordinate frame whose origin is usually located on the take off point of the quadrotor.

2. **Planner** area of research.

   This research area is in charge of selecting what the quadrotor has to do. It can generate either $3D$ points or velocities or trajectories. The quadrotor will make decision based on what the user wants or on how the quadrotor interacts with the environment by means of sensors. Vision based planners can be developed aiming at solving object following and autonomous landing tasks using visual feedback to generate reference velocities that the quadrotor has to track to accomplish the desired task.

[1] http://www.vision4uav.eu
[2] https://github.com/Vision4UAV/Aerostack/wiki

3. **Control** area of research.

   This research field is mainly focus on how the quadrotor can track the reference given by the planner (position, velocity, orientation, trajectory), minimizing the error between the desired references provided by the planner and the estimated ones given by the state estimator.

A scheme summarizing how the three researches area cooperate to complete an autonomous mission is provided.

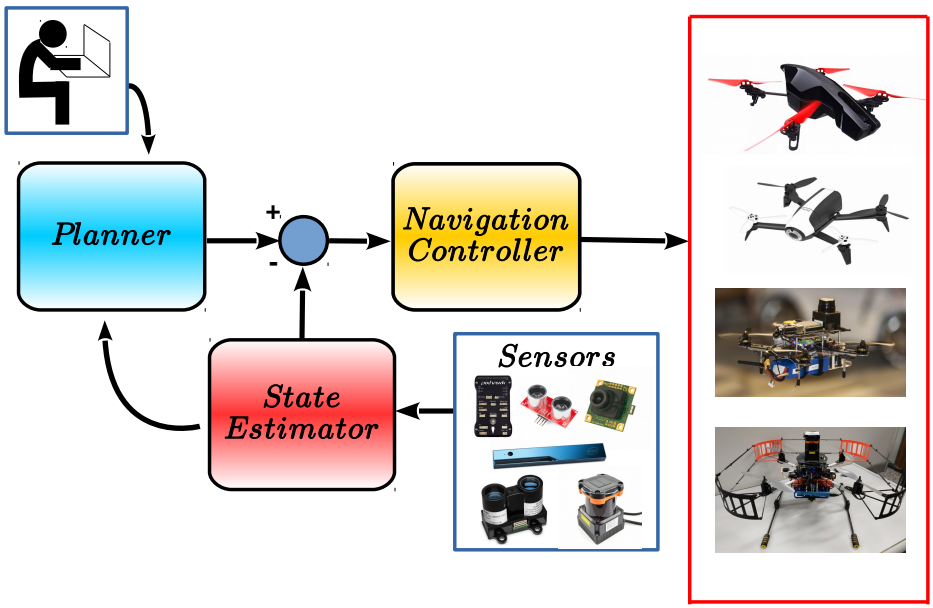


Figure 1-1: Combination of the three areas of research to achieve fully autonomous quadrotor flight

To summarize, the planner receives as a input the estimation provided by the state estimator and it generates as a output the desired reference commands required to accomplish the planned task. The navigation controller takes as inputs both the planner and the state estimator outputs and it uses them to generate specific controller outputs for the quadrotor. Applying the controller outputs to the quadrotor the latter moves in a way that the desired reference commands are tracked minimizing the error between the latter and the estimated ones. When the error is small enough the planner sends new reference commands an a new task starts.

1-2 Problem description

To achieve autonomous quadrotor navigation in both indoor and outdoor environment two different type of approaches can be used.

1. **Pose based approach**.

   This approach consists in sending to the quadrotor the desired pose (position and orientation) that it has to reach expressed in world coordinate frame. If obstacles are present in the environment the quadrotor has to be able to plan a trajectory able to avoid obstacles minimizing the flight time required to move from the initial to the final desired location. The design of the trajectory planner belongs to a second research area that is the planner area. To be able to move from the initial pose to the desired one the quadrotor requires to know at every time instant its current pose which it is estimated combining information coming from the quadrotor's sensors. The reference pose is expressed in world coordinate frame and it is sent to the quadrotor. The latter moves until the error between the desired and measured one is driven to zero. A navigation controller framework is required to ensure that the desired references are tracked correctly.

   Advantages

   - It does not require visual information to achieve the task. To recognize an object a detector is required and to speed up the detection frame rate a tracker is used. The detector is computationally expensive and to detect something it requires the object to be always visible in the image which might not be the case if the desired location can be far from the initial take off point.

   Disadvantages

   - It requires to know the position of the quadrotor in world coordinate frame. The latter in outdoor environments can be retrieved from GPS data but in indoor environment it requires different sensors to be fused together to achieve a proper result.
   - It requires to know the desired quadrotor's pose. This is usally not a problem because the latter is chosen a priori. However it becomes a problem when the quadrotor either has to follow a moving object whose pose is unknown in space or it has to land on a moving platform whose pose might change randomly.

2. **Vision based approach**.

   This approach uses visual information as a feedback directly in the control law to compute the desired velocities expressed in world coordinate frame that the quadrotor has to track to located itself at a desired distance from the chosen object. The quadrotor is not aware of it is pose and it moves with respect to a chosen desired object whose pose can change in time. The vision based approach is computed combining the vision based planner with the navigation controller framework. The main difference from the pose based approach is that the vision based approach use visual information to compute the desired velocities (not position) required to drive asymptotically the quadrotor up to a certain desired distance from the detected object.

   Advantages

- It does not require to know the position of the quadrotor with respect to a world fixed coordinate frame.
- There is no need to know a priori the object position in world coordinate frame. This is a great advantage because it allows the quadrotor to move with respect to an object whose position can change in time without the need to add external sensors to calculate the pose of the desired object.

Disadvantages

- A visual marker is required to be placed on the desired location to be reached and the visual marker needs to be always visible in the acquired image. Nowadays, machine learning power is growing and the need to add visual marker to identify a specific object is not required anymore. Powerful algorithms have been developed in the state of the art to be able to directly identify objects.

Both the approaches have been used to solve autonomous quadrotor navigation. The goal of this thesis is to provide a framework in which it is possible to easily switch between the two approaches to solve different type of missions. The idea is that it has to be simple to generate a mission in which firstly the quadrotor reaches a first predefined pose, secondly it follows an object only when it appears on the acquired camera image, thirdly it makes a square given four different predefined desired pose and finally it autonomously lands either on a static or moving platform.

1-3 Goals

The goal of this thesis is to achieve autonomous quadrotor navigation either relying on position measurements (pose based approach) or visual information (vision based approach). The final goals of this thesis are

- ***Design a navigation controller framework able to communicate with both parrot AR Drone 2.0 and Pixhawk autopilot that it is able to track simultaneously either the quadrotor's desired positions and the yaw angle*** $(x^\star, y^\star, z^\star, \psi^\star)$ ***or the desired velocities and the yaw angle*** $(\dot{x}^\star, \dot{y}^\star, \dot{z}^\star, \psi^\star)$ ***or the horizontal desired velocities, the altitude and the yaw angle*** $(\dot{x}^\star, \dot{y}^\star, z^\star, \psi^\star)$***.***
- ***Design a vision based planner that uses visual information acquired either by the front or by the bottom camera to calculate the desired translational velocities*** $(\dot{x}^\star, \dot{y}^\star, \dot{z}^\star)$ ***required by the quadrotor to approach up to a desired distance a chosen static or moving object keeping the latter centered in the acquired camera image without knowing neither the quadrotor's pose nor the object one. The translational velocities are sent to the navigation controller framework. The latter ensures that the desired velocities are tracked correctly.***

Assuming to have an estimate of the position provided by the EKF the navigation controller framework ensures that given a desired pose (position and orientation) and an estimation of

it the quadrotor moves from the current to the desired one. The vision based planner allows to face tasks as object following and autonomous landing and it works without the need to estimate the quadrotor's pose.

1-4 Approaches

To achieve the desired goals

1. A cascade control design made up by seven 2DOF PID controllers divided into five different modules (horizontal position controller, vertical position controller, horizontal speed controller, vertical speed controller, yaw controller) has been developed to ensure that the navigation controller framework is able to track simultaneously either $(x^\star, y^\star, z^\star, \psi^\star)$ or $(\dot{x}^\star, \dot{y}^\star, \dot{z}^\star, \psi^\star)$ or $(\dot{x}^\star, \dot{y}^\star, z^\star, \psi^\star)$. An explanation is provided on why this design has been chosen for the navigation controller framework.

2. Three different modules (perception, image state estimator, image based visual servo) has been developed to achieve the vision based planner objective. Planar ArUco markers have been used to avoid the need of designing different specific objects' detectors. Their corners have been extracted by the ArUco detector located inside the perception module. An image state estimator module has been developed inside which a Kalman filter with velocity constant model has been designed to estimate the visual marker's corners when a detection is suddenly lost. Given an estimation of the visual marker corners at a specific chosen frequency an image based visual servo algorithm embedded inside the image based visual servo module has been developed to calculate the desired quadrotor's translational velocities $(\dot{x}^\star, \dot{y}^\star, \dot{z}^\star)$ required by the navigation controller framework to minimize the error between the estimated and the desired corners where the latter are computed as a function of the desired distance between the quadrotor and the visual marker.

To validate both the navigation controller framework and the vision based planner firstly the navigation controller has been validated separately firstly in simulation using Gazebo simulator in combination with PX4 Software-In-The-Loop and secondly in real flight with the Parrot AR Drone 2.0. After having adjusted the control parameters, the vision based planner has been combined with the navigation controller framework to solve high level tasks such as object following and autonomous landing either on static or moving platform. In particular a mission has been chosen in which firstly the quadrotor autonomously reaches a first predefined desired pose, secondly it approaches up to a desired distance a static visual marker using the front camera image data, thirdly it reaches a second predefined desired pose and finally it autonomously lands using the bottom camera image information either on a static or moving visual marker. The mission has been tested in real flight with Parrot AR. Drone 2.0 and in simulation using Gazebo simulator in combination with PX4 Software-In-The-Loop. All the modules have been developed to be in tune with the Aerostack conventions. To conclude, Aerostack project enforces modularity and this is achieved using the open source software operating system Ubuntu 16.04.2 LTS (Xenial Xerus)[3] , ROS Kinetic Kame[4] as

[3] http://releases.ubuntu.com/16.04/
[4] http://wiki.ros.org/kinetic

robotic middleware to ensure system communication and Gazebo 7 as a simulator that comes automatically installing ROS Kinetic Kame. Thus, in order make the code compatible compatible with the Aerostack the same version of Ubuntu, ROS and Gazebo will be chosen. Furthermore, the algorithms will be coded using as a programming language $C++$ following the ROS $C++$ style guide[5] to make the code readable by a third person and easy to be debugged.

[5] http://wiki.ros.org/CppStyleGuide

Chapter 2

Navigation controller framework

In this chapter the main features of the design of the navigation framework are presented. An illustration of the navigation controller framework is provided.

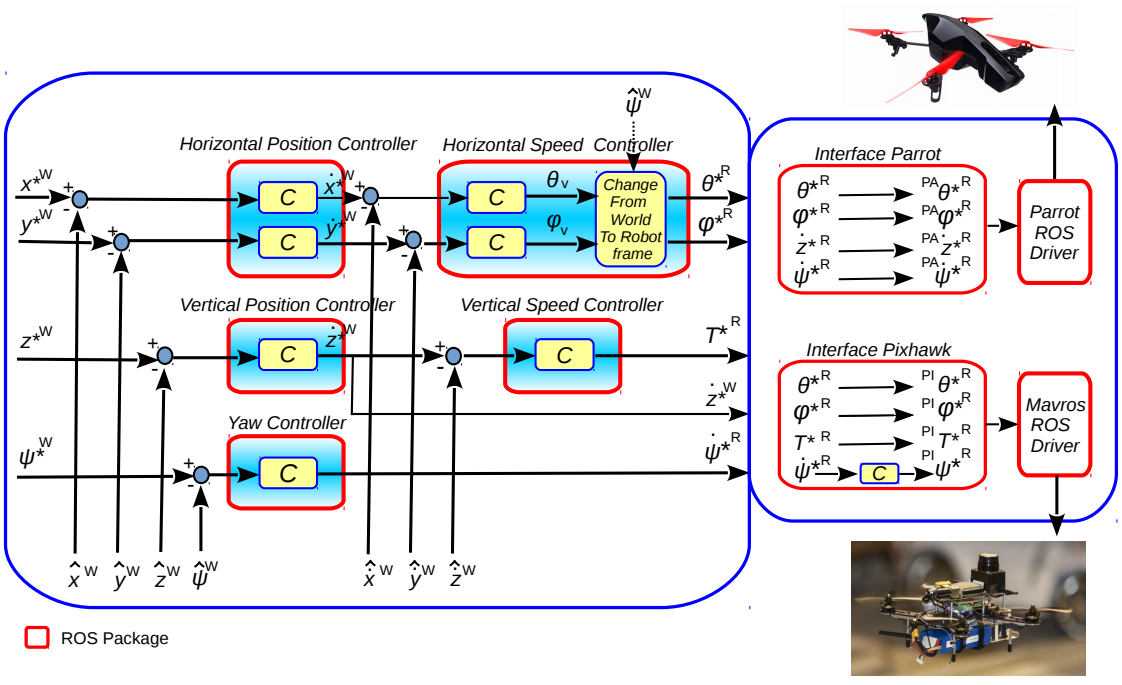


Figure 2-1: Navigation controller framework to control both quadrotor equipped with Parrot and Pixhawk autopilot

The latter has been designed with the aim to improve the Aerostack navigation controller framework described in [3]. The reason why it has been chosen to design the controller framework as showed in Figure 2-1 are summarized as follows.

1. **Modular design:** The design has been inspired by the Matlab Simulink approach. In the latter the controller are represented as blocks. According to the controller type the same block can be used multiple times simply modifying the default parameters of the block. In the controller framework design the approach it is similar. Indeed the controllers are embedded into a library. The library is used inside a module which runs at a chosen frequency and it is in charge of getting inputs an send outputs data. The module uses the controller available in the controller library to generate the desired outputs. Having a framework like this facilitate the develop and test of new controllers. Indeed if a new controller is designed it is required to add it inside the controller library and inside the chosen module it is possible to replace the previous used one with the new desired one. In doing this it is possible to take advantage of the fact that the data required by the module to accomplish the task are not modified. What it is going to change inside a module is the controller chosen from the controller library to solve the task. Different modules will use the same controller library with different input data which are the ones required by the chosen module to accomplish the task.

2. **Module activation and deactivation capability:** Each module can be activated or deactivated. This ensure to easily switch between position control to velocity control simply shutting down the horizontal and vertical position control. The process of shutting down means that the module will not get any data and will not send any output anymore. However, it is able to listen to the activation or deactivation variable used to activate or deactivate it.

3. **Modules structure design:** The choice to split the framework into yaw, horizontal position, vertical position, horizontal speed and vertical speed controller is related to the task that a quadrotor usually performs. Indeed a quadrotor can freely move in space receiving a desired $(3D)$ input position and yaw angle. In doing this all the controllers inside the framework have to be activated. However, a quadrotor using the bottom camera can attempt to located itself on a bucket to release an object on it. To do this the quadrotor can be controller in the velocity domain which means that it does not required the horizontal position controller to be activated. Furthermore, it is possible to choose to control the altitude or the velocity along the (z) direction to make the quadrotor able to release an object inside the bucket. Both the choices are available because it is possible to choose to activate or deactivate the vertical position controller. The choice to split the horizontal and vertical speed controllers is connected to the fact that the Parrot Autopilot does not required the vertical speed controller to be activated. The latter is already embedded into the Parrot Autopilot itself. On the other hand this module is required in dealing with the Pixhawk Autopilot. For this reason it has been chosen to split the horizontal and vertical speed controller. Indeed, if the navigation controller is used to control a Parrot quadrotor the navigation controller framework will have the horizontal position, the vertical position , the horizontal speed , the vertical speed and the yaw controller activated. In trying to control a quadrotor equipped with Pixhawk Autopilot also the vertical speed controller will be activated.

4. **Modules frequency choice:** Each module can run at a chosen frequency which depends on the input frequency data and on the cascade control design choice. Indeed, according to the cascade theory is required that the horizontal and vertical position controller will run at a frequency lower than the horizontal and vertical speed controller. The input data of the horizontal and speed controller are calculated by an EKF in charge of estimating the full quadrotor' states (position, velocity, acceleration, orientation) expressed in world coordinate fusing together different sensors information. Thus the frequency of the speed controller has to be set to be the same of the frequency at which the EKF provides an estimation of the desired input. After having selected the frequency of the horizontal and speed controller modules the frequency of the horizontal and vertical position modules is set to be lower than the frequency of the position controllers modules. To define how much lower the frequency of the position controller modules has to be with respect to the frequency of the speed controllers module firstly it is required to tune the speed controller modules ensuring that they are able to track the desired reference velocities, secondly the position controllers modules are added to try to track a certain reference position at a certain frequency from three to ten times lower than the speed controller modules frequency. In checking the velocities outputs with respect to the references it is required that the speed controller modules will have enough time to reach the reference velocities provided by the position controller modules. If this does not occur it is required to reduce the frequency of the position controller modules. Furthermore, another factor which influences the choice of the frequency of the position controller modules is the frequency at which an estimation of the quadrotor's pose (position and orientation) is available. Indeed it is possible that the estimation of the quadrotor's pose is perform at a frequency lower than the estimation of the quadrotor's velocities and acceleration. Two different EKF fusing different data might be developed to solve this two different estimation problems. Therefore the position controller modules frequency has to be set lower or equal than the frequency at which the estimation of the quadrotor's pose is provided. If the quadrotor's pose estimation is provided at the same frequency of the quadrotor's velocities and accelerations estimation a rule of thumb is to set the position controller modules frequency around three of four times lower than the frequency of the speed controller modules.

5. **Inputs and outputs navigation controller framework convention:**

Table 2-1: Table summarizing the frame convention used to expressed the inputs and the outputs of the navigation controller framework

Data	Convention (C)	Description
$x^{\star W}$ $y^{\star W}$	ENU convention: unit $[m]$ $x^{\star W} \rightarrow$ (+ forward, − backward) $y^{\star W} \rightarrow$ (+ leftward, − rightward)	Horizontal positions references expressed in world coordinate frame. $x^{\star W}$, $y^{\star W}$ are the inputs of the horizontal position controller.
$z^{\star W}$	ENU convention: unit $[m]$ $z^{\star W} \rightarrow$ (+ upward, − downward)	Vertical position reference expressed in world coordinate frame. $z^{\star W}$ is the input of the vertical position controller.
$\psi^{\star W}$	ENU convention: unit $[rad]$ $\psi^{\star W} \rightarrow$ (+ ccw, − cw) $\psi^{\star W} \in \begin{bmatrix} -\pi & \pi \end{bmatrix}$	Reference yaw angle expressed in world coordinate frame. $\psi^{\star W}$ is the input of the yaw controller.
$\dot{x}^{\star W}$ $\dot{y}^{\star W}$	ENU convention: unit $[m/s]$ $\dot{x}^{\star W} \rightarrow$ (+ forward, − backward) $\dot{y}^{\star W} \rightarrow$ (+ leftward, − rightward) $\dot{x}^{\star W}, \dot{y}^{\star W} \in \begin{bmatrix} -1 & 1 \end{bmatrix}$	Horizontal velocities references expressed in world coordinate frame. $\dot{x}^{\star W}, \dot{y}^{\star W}$ are the inputs of the horizontal speed controller or the outputs of the horizontal position controller.
$\theta^{\star R}$ $\phi^{\star R}$	ENU convention: unit $[rad]$ $\theta^{\star R} \rightarrow$ (+ forward, − backward) $\phi^{\star R}$ (+ rightward, − leftward) $\theta^{\star R}$, $\phi^{\star R} \in \begin{bmatrix} -0.2828 & 0.2828 \end{bmatrix}$	Reference pitch $\theta^{\star R}$ and roll $\phi^{\star R}$ expressed in robot coordinate frame. $\theta^{\star R}$, $\phi^{\star R}$ are the outputs of the horizontal speed controller.
$\dot{z}^{\star R}$	ENU convention: unit $[m/s]$ $\dot{z}^{\star R} \rightarrow$ (+ upward, − downward) $\dot{z}^{\star} \in \begin{bmatrix} -1 & 1 \end{bmatrix}$	Vertical velocity reference expressed in robot coordinate frame. $\dot{z}^{\star R}$ is the output of the vertical position controller or the input of the vertical speed controller. Small angle approximation ensures that $z^{\star R} \approx z^{\star W}$.
$T^{\star R}$	ENU convention: dimensionless $T^{\star R} \rightarrow$ (+ upward, − downward) $T^{\star R} \in \begin{bmatrix} 0 & 1 \end{bmatrix}$	Reference thrust expressed in robot coordinate frame. The thrust required by the Pixhawk autopilot is in between zero and one therefore no measurement thrust unit has been used. $T^{\star R}$ is the output of the vertical speed controller. Small angle approximation ensures that $T^{\star R} \approx T^{\star W}$.
$\dot{\psi}^{\star R}$	ENU convention: unit $[rad/s]$ $\dot{\psi}^{\star R} \rightarrow$ (+ ccw, − cw) $\dot{\psi}^{\star R} \in \begin{bmatrix} -0.4 & 0.4 \end{bmatrix} \approx \pm 23°/s$	Reference yaw angular velocity expressed in robot coordinate frame. $\dot{\psi}^{\star R}$ is the output of the yaw controller. Small angle approximation ensures that $z^{\star R} \approx z^{\star W} \implies \dot{\psi}^{\star R} \approx \dot{\psi}^{\star W}$.
$\hat{x}^{W}$ $\hat{y}^{W}$ $\hat{\psi}^{W}$	ENU convention: unit $[m]$ $\hat{x}^{W} \rightarrow$ (+ forward, − backward) $\hat{y}^{W} \rightarrow$ (+ leftward, − rightward)	Estimated horizontal positions $\hat{x}^{W}$, $\hat{y}^{W}$ expressed in world coordinate frame. $\hat{x}^{W}$, $\hat{y}^{W}$ are the outputs either of EKF or MOCAP.
$\hat{z}^{W}$	ENU convention: unit $[m]$ $\hat{z}^{W} \rightarrow$ (+ upward, − downward)	Estimated vertical position $\hat{z}^{W}$ expressed in world coordinate frame. $\hat{z}^{W}$ is the output either of EKF or MOCAP.
$\hat{\psi}^{W}$	ENU convention: unit $[rad]$ $\hat{\psi}^{W} \rightarrow$ (+ ccw, − cw) $\hat{\psi}^{W} \in \begin{bmatrix} -\pi & \pi \end{bmatrix}$	Estimated yaw angle $\hat{\psi}^{W}$ expressed in world coordinate frame. $\hat{\psi}^{W}$ is the output either of EKF or MOCAP.
$\hat{\dot{x}}^{W}$ $\hat{\dot{y}}^{W}$	ENU convention: unit $[m/s]$ $\hat{\dot{x}}^{W} \rightarrow$ (+ forward, − backward) $\hat{\dot{y}}^{W} \rightarrow$ (+ leftward, − rightward)	Estimated horizontal velocities expressed in world coordinate frame. $\hat{\dot{x}}^{W}$, $\hat{\dot{y}}^{W}$ are the outputs either of EKF or MOCAP.
$\hat{\dot{z}}^{W}$	ENU convention: unit $[m/s]$ $\hat{\dot{z}}^{W} \rightarrow$ (+ upward, − downward)	Estimated vertical velocity expressed in world coordinate frame. $\hat{\dot{z}}^{W}$ is the output either of EKF or MOCAP.

A figure underlining the convention used for the inputs and outputs data of the navigation controller framework is provided.

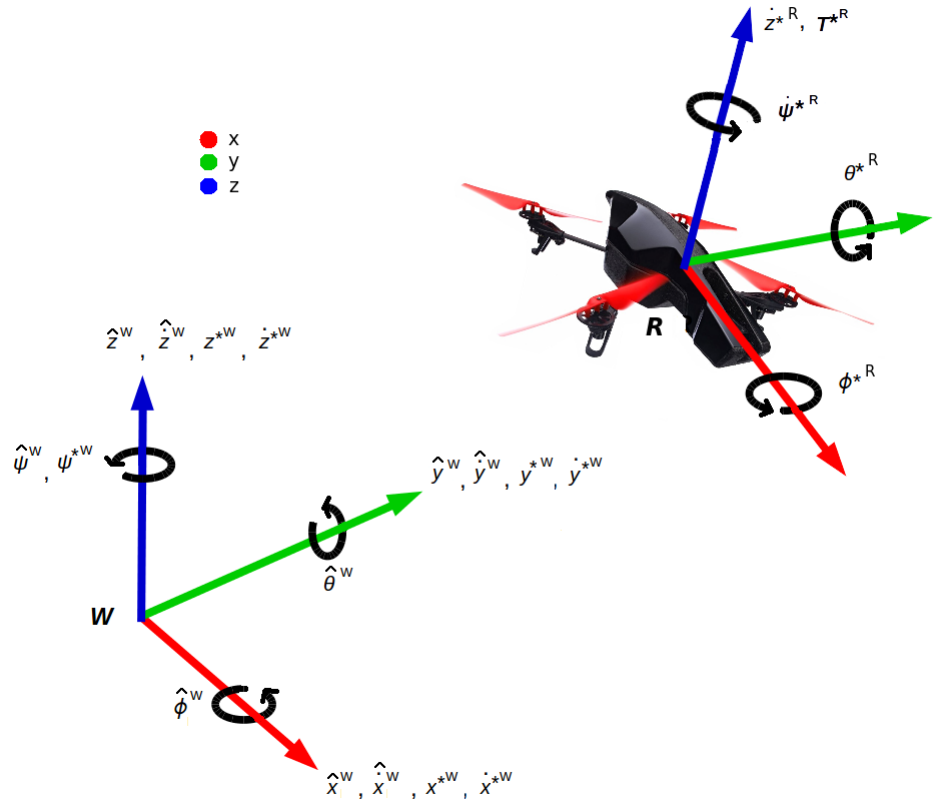


Figure 2-2: Navigation controller framework references $\left(x^{\star W},\ y^{\star W},\ z^{\star W},\ \dot{x}^{\star W},\ \dot{y}^{\star W},\ \dot{z}^{\star W}\ \psi^{\star W}\right)$, EKF or MOCAP feedbacks $\left(\hat{x}^{W},\ \hat{y}^{W},\ \hat{z}^{W},\ \hat{\psi}^{W},\ \hat{\dot{x}}^{W},\ \hat{\dot{y}}^{W},\ \hat{\dot{z}}^{W}\ \right)$ and navigation controller framework controller outputs $\left(\theta^{\star R},\ \phi^{\star R},\ \dot{z}^{\star R},\ \dot{\psi}^{\star R},\ T^{\star R}\right)$. The controller references and the feedbacks are expressed in world (W) fixed coordinate frame whereas the controller outputs are in robot (R) coordinate frame.

2-1 Two degree of freedom PID controller

In this section the parallel 2DOF PID single input single output controller used in all the navigation controller framework module is presented. The controller output is given in both the Laplace and discrete Z domain. The difference equation used to calculate the controller output is also provided to simplify the code implementation of the algorithm. The choice to develop a 2DOF PID controller is motivated by the fact that this controller allows to easily implement different PID controller structures. Indeed it is possible to say that among the different PID controller designs' type it is the most generic one. Indeed, it is a type of PID controller where it is possible to weight both the proportional and the derivative error by means of two tuning parameters (b, c). The structure it is shown in the following figure.

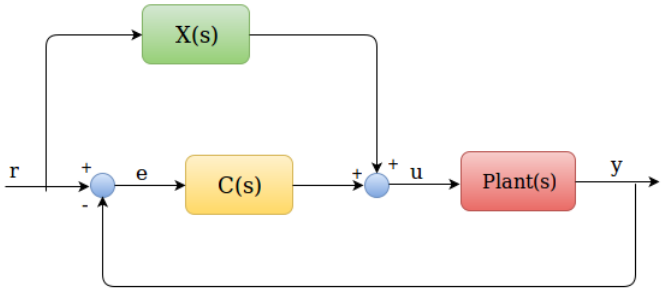


Figure 2-3: 2DOF PID Controller block diagram

The 2DOF PID controller structure is the result of the composition of a feedback ($C(s)$) and a feed-forward controller ($X(s)$. The feedback controller transfer function in the Laplace domain is defined as

$$C(s) = U(s)/E(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + sK_d \quad E(s) = R(s) - Y(s) \tag{2-1}$$

whereas the Laplace transfer function of the feed-forward controller is given by

$$X(s) = \frac{U(s)}{R(s)} = (b-1)K_p + (c-1)sK_d \tag{2-2}$$

Thus the feedback controller represents a simple PID controller whereas the feed-froward controller stands for a PD controller. Thus it is possible to conclude that the 2DOF PID controller is the combination of a feedback PID and a feed-forward PD controller. The 2DOF PID controller output ($U(s)$) in the Laplace domain is given by the sum of both the feedback and the feed-forward controller outputs.
The Laplace feedback controller output is defined as

$$\begin{aligned} U(s) = C(s)E(s) = K_pE(s) + \frac{K_iE(s)}{s} + sK_dE(s) = \\ = K_pR(s) - K_pY(s) + \frac{K_iR(s)}{s} - \frac{K_iY(s)}{s} + sK_dR(s) - sK_dY(s) \end{aligned} \tag{2-3}$$

whereas the Laplace feedforward controller output is

$$\begin{aligned} U(s) = X(s)R(s) = (b-1)K_pR(s) + (c-1)sK_dR(s) = \\ = bK_pR(s) - K_pR(s) + csK_dR(s) - sK_dR(s) \end{aligned} \tag{2-4}$$

The 2DOF PID controller output in the Laplace domain is calculated as

$$\begin{aligned} U(s) = C(s)E(s) + X(s)R(s) = K_pR(s) - K_pY(s) + \frac{K_iR(s)}{s} - \frac{K_iY(s)}{s} + \\ + sK_dR(s) - sK_dY(s) + bK_pR(s) - K_pR(s) + csK_dR(s) - sK_dR(s) = \\ = K_p\Big(bR(s) - Y(s)\Big) + \frac{K_i}{s}\Big(R(s) - Y(s)\Big) + sK_d\Big(cR(s) - Y(s)\Big) = \\ = K_pE_p(s) + \frac{K_i}{s}E_i(s) + sK_dE_d(s) = U_p(s) + U_i(s) + U_d(s) \end{aligned} \tag{2-5}$$

where (U_p, U_i, U_d) represent respectively the proportional,integral and derivative controller outputs actions performed by 2DOF PID controller. In the following subsection an explanation of the input of the controller is provided together with the controller output derivation. In the last subsection the 2DOF PID controller algorithm is presented.

2-1-1 Inputs

1. $\mathbf{T_s}$: It represents the sampling time used in the design of the 2DOF PID controller. The sampling time choice is usually calculated as the inverse of the frequency at which the measurement is provided. This choice ensures that every time the 2DOF PID runs the reference and the measurement data are available to calculate the controller output.

2. $\mathbf{K_p}, \mathbf{K_d}, \mathbf{K_i}$: They represents respectively the proportional, derivative and integral gains of the 2DOF PID controller. The tuning of these parameters is a well known problem in the state of the art. For the sake of this thesis it has been chosen to use the Matlab PID Tuner[1] to retrieve the controller parameters. To do this a discrete model derived analyzing the quadrotor dynamics is used. Given the discrete model the PID Matlab Tuner has been used to tune the controllers parameters such that steady state error is achieved and the controller output saturation boundaries are satisfied. The choice to use the Matlab Tuner is related to the fact that the aim of the navigation controller framework is to be able to work with different quadrotors providing a fast way to tune the parameters according to the needed purpose. Having a 2DOF PID controller has also the advantage that is possible to play with the weight parameters (b, c) to ensure that the measurement (plant output) does not have any overshoot. This is a good advantage in dealing with quadrotor because it allows the quadrotor to reach $(3D)$ position close to obstacles without having any overshoot (over-damped behavior). In tuning the controllers a filter on the derivative error term has been introduced with the purpose to make the control sensitivity function realizable. The latter allows to see given a certain input what is the controller output (when the control loop it is closed). Knowing the controller output the parameters have been adjusted to keep the controller output inside the desired saturation boundaries.

3. $\mathbf{b}, \mathbf{c}$: They are two weighted parameters $(b, c \in \begin{bmatrix} 0 & 1 \end{bmatrix})$ used to weight the proportional and the derivative error terms. In combining these parameters in a different ways different PID controller structures can be selected [4]. Given the 2DOF PID controller output presented in Eq. (2-5) different types of PID designs can be implemented modifying the weight parameters (b, c).

Table 2-2: PID Controller Structures

	b	c	Laplace Equation
1)PID	1	1	$U(s) = K_p\Big(R(s)-Y(s)\Big)+\frac{K_i}{s}\Big(R(s)-Y(s)\Big)+sK_d\Big(R(s)-Y(s)\Big)$
2)PI-D	1	0	$U(s) = K_p\Big(R(s)-Y(s)\Big)+\frac{K_i}{s}\Big(R(s)-Y(s)\Big)+sK_d\Big(-Y(s)\Big)$
3)I-PD	0	0	$U(s) = K_p\Big(-Y(s)\Big)+\frac{K_i}{s}\Big(R(s)-Y(s)\Big)+sK_d\Big(-Y(s)\Big)$
4)ID-P	0	1	$U(s) = K_p\Big(-Y(s)\Big)+\frac{K_i}{s}\Big(R(s)-Y(s)\Big)+sK_d\Big(R(s)-Y(s)\Big)$

The PID number one has the drawback that the derivative term contains the set-point $(R(s))$ value. Thus, a sudden change in the set-point might cause an unwanted large change in the derivative controller output action. To solve this problem the PID number

[1] https://es.mathworks.com/help/control/pid-controller-design.html

two can be chosen. The latter does not use the reference to calculated the derivative error term and for this reason the derivative controller output action when a step reference is applied does not present any spikes that instead appear using the PID number one. To conclude decreasing the value of (b) the overshoot of the system is reduced but the system response becomes slower. On the other hand decreasing the parameter (c) close to zero the controller output action is reduced ensuring a smooth set point tracking (over-damped behavior)). Thus using a 2DOF PID controller is possible to modify the weights according to the requirements (rise time, overshoot, settling time, steady state error, disturbance rejection) that the system has to satisfy.

4. $\mathbf{U_{Min}}, \mathbf{U_{Max}}$: They represent the saturation constraints applied to the controller output. In dealing with quadrotors the 2DOF PID controller output has a physical value. Thus it is important to properly saturate this value according to the physical meaning and unit of measure that it has.

5. $\mathbf{enable_{ref_{filter}}}, \mathbf{N_r}$: These parameters are introduced with the aim to be able to filter the supplied reference value sent to the 2DOF PID controller. A low pass filter with filter gain (N_r) is used to filter the reference value. The variable ($enable_{ref_{filter}}$) is a boolean variable that if it is true it will ensure that the supplied reference is filtered with a low pass filter with filter gain equal to (N_r). The latter determines how fast the set-point reference changes. The low pass filter Laplace transfer function between the reference input ($R(s)$) and the filtered reference output ($R_f(s)$) is given by

$$R_f(s) = \frac{N_r}{N_r + s} R(s) \tag{2-6}$$

where ($R_f(s)$) and ($R(s)$) stand for respectively the filtered reference and supplied reference values in the Laplace domain. However in designing an algorithm able to perform this filter operation is required to calculate the difference equation relating the filtered reference with the supplied one. To do this firstly Eq. (2-6) is discretized ((Z) domain) using Backward Euler discretization method ($s \approx \frac{z-1}{T_s}$) as follows

$$R_f(z) = \frac{1}{1 + N_r T_s} R_f(z) z^{-1} + \frac{N_r T_s}{1 + N_r T_s} R(z) \tag{2-7}$$

secondly the difference equation is derived

$$r_f(k) = \frac{1}{1 + N_r T_s} r_f(k-1) + \frac{N_r T_s}{1 + N_r T_s} r(k) \tag{2-8}$$

Knowing the difference equation and the sampling time (T_s) representing the inverse of the frequency at which the controller is running it is possible to calculate given a supplied reference ($r(k)$) at time instant (k) what is the value of the filtered reference ($r_f(k)$) at the same time instant (k). To summarize, the boolean value of ($enable_{ref_{filter}}$) determines if the reference has to be filtered or not. The choice to introduce a filter applied to the reference is related to the need to avoid spikes on the 2DOF PID controller output. The latter occurs mainly when a step reference is provided. Indeed, given the PID derivative controller output action

$$u_d(k) = \frac{K_d}{T_s}(e_d(k) - e_d(k-1) \quad with \quad e(k) = cr(k) - y(k) \tag{2-9}$$

if a step reference is supplied to the 2DOF PID controller a large difference will occur between $(e_d(k))$ and $(e_d(k-1)$ which divided by the sampling time will result in large derivative controller output action. Introducing a filter on the supplied reference will result in a smother supplied reference that will cause a smaller derivative controller action. To tune the filter gain (N_r) is better to plot the filter output and the supplied reference given different filter gain (N_r) values to see how much the filter gain affects the supplied reference. However, a rule of thumb to choose (N_r) is to ensure that the sampling time (T_s) is at least five times smaller than the filter time constant $(1/N_r)$.

$$T_s \leq \frac{(\frac{1}{N_r})}{5} \rightarrow N_r \leq \frac{1}{5T_s} \tag{2-10}$$

6. $\mathbf{enable_{der_{filter}}}, \mathbf{N_d}$: They are parameters that have been introduced to filter the derivative error term. The boolean variable is used to enable $(enable_{der_{filter}} = True)$ or not $(enable_{der_{filter}} = False)$ the derivative filter applied to the derivative error term. The choice to apply a low pass filter on the derivative error term has the aim to ensure that the control sensitivity function is physical realizable. The latter is defined as

$$\frac{U(s)}{R(s)} = \frac{C(s)}{1 + G(s)C(s)}$$

where $R(s)$, $(U(s))$, $(C(s))$ and $(G(s))$ are respectively the reference supplied to the system, the controller output action, the PID controller transfer function and the plant transfer function. Only when the control sensitivity transfer function is physically realizable it is possible to see what is the controller output action $(U(s))$ associated to a specific reference supplied input $(R(s))$. In dealing with a quadrotor it is really helpful to have the possibility to tune the 2DOF PID controller such that given a certain reference the generated controller output is bounded inside some predefined saturation values. For example inside the horizontal position controller module two single input single output 2DOF PID controllers are located. The goal of them is to ensure that given the desired (x) and (y) quadrotor's position expressed in world coordinate frame the quadrotor will track them. Because a cascade control design has been developed in the navigation controller framework the output of the horizontal position controllers are the desired velocities at which the quadrotor has to move along the (x) and (y) direction of the world coordinate frame. These velocities are physical values and to ensure that the quadrotor will not move too fast they are bounded between one and minus one meter per second. Having a realizable control sensitivity function allows during the tuning of the controllers to have the possibility to see what are the horizontal position controllers outputs given the supplied inputs (when the control loop it is closed). Thanks to this information it is possible to check if the controller outputs are inside the desired boundaries or not. If this does not occurs it is required to change the tuning controllers parameters. The low pass filter transfer function in the Laplace domain is

$$E_{d_f}(s) = \frac{N_d}{N_d + s} E_d(s) \tag{2-11}$$

where $(E_{d_f}(s))$ and $(E_d(s) = cR(s) - Y(s))$ are respectively the filtered and not filtered derivative error term of the 2DOF PID controller. Using Backward Euler discretization

method ($s \approx \frac{z-1}{T_s}$) Eq. (2-11) becomes

$$E_{d_f}(z) = \frac{1}{1 + N_d T_s} E_{df_f}(z) z^{-1} + \frac{N_d T_s}{1 + N_d T_s} E_d(z) \tag{2-12}$$

The difference equation associated to Eq. (2-12) is provided.

$$e_{d_f}(k) = \frac{1}{1 + N_d T_s} e_{d_f}(k-1) + \frac{N_d T_s}{1 + N_d T_s} e_d(k) \tag{2-13}$$

Eq. (2-10) provided a rule of thumb to tune the filter gain (N_d) knowing the sampling time (T_s). To conclude, only if the boolean variable ($enable_{der_{filter}}$) is set to true the low pass filter, with filter gain (N_d), is applied on the derivative error term.

7. $\mathbf{enable_{anti_{windup}}}, \mathbf{K_{aw}}$: These two variables represent respectively a boolean variable used to enable or disable the anti wind up scheme and the gain associate to the anti wind up back calculation approach. In dealing with small angle approximation assumption it is important to properly choose the saturation boundaries associated to the controllers' outputs. A problem related to the controller output saturation that occurs only if an integrator is present and the controller output saturate is the wind up. When this occurs the integrator error continues to increase leading the controller output value to increase as well over the saturated maximum value. This causes an undesired behavior in the output of the plant due to the fact that the controller output provided to the plant is the saturated one and not the one provided by the 2DOF PID controller. If this situation occurs a constant error will appear between the reference and the measurement plant values which will introduce lag in the system. To solve the wind up problem a back calculation technique has been applied. The latter consists in adding to the PID integrator another integrator multiplied times the saturated error ($U_{sat} - U$). A block diagram is shown.

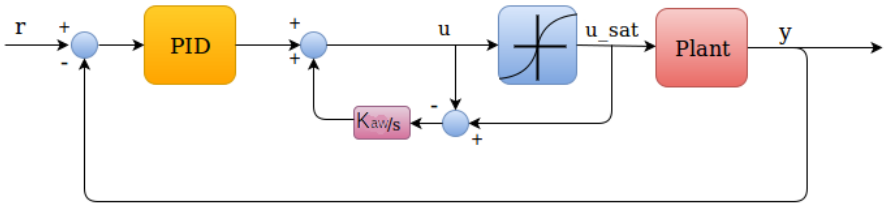


Figure 2-4: Anti wind up using back calculation block diagram

Given Figure 2-4 is possible to infer that the expression in the Laplace domain of integrator error term introduced to solve wind up problem is

$$U_a w(s) = \frac{K_{aw}}{s} E_s at(s) \quad with \quad E_{sat}(s) = (U_{sat}(s) - U(s) \tag{2-14}$$

where ($U_{sat}(s)$) and ($U(s)$) are respectively the saturated and not saturated controller outputs. (K_{aw}) represents the integrator tunable gain used to reduce the integrator error when saturation occurs . Applying Backward Euler discretization method ($s \approx \frac{z-1}{T_s}$) Eq. (2-14) becomes

$$U_a w(z) = \frac{T_s}{1 - z^{-1}} K_{aw} E_{sat}(z) \tag{2-15}$$

The difference equation associated to (2-15) is

$$u_{aw}(k) = u_{aw}(k-1) + T_s K_{aw} e_{sat}(k) \tag{2-16}$$

with $(e_{sat}k = u_{sat}(k) - u(k))$.

2-1-2 Outputs

1. **$\underline{\mathbf{U_{sat}}}$**: It represents the controller output saturated value calculated by the 2DOF PID controller. To derive this value it is required to calculate the difference 2DOF PID controller equation. To do this firstly the 2DOF PID controller is derived in the Laplace domain. Secondly Backward Euler discretization approach is applied to retrieve the controller output equation in the (Z) domain. Finally the difference equation is presented. Given the Laplace equation of the 2DOF PID controller showed in Eq. (2-5) applying Backward Euler discretization method $(s \approx \frac{z-1}{zT_s})$ the latter becomes

$$U(z) = U_p(z) + U_i(z) + U_d(z) \tag{2-17}$$

with the proportional, integral derivative controller output actions $(U_p(z), U_i(z), U_d(z))$ equal to

$$\begin{aligned} U_p(z) &= K_p E_p(z) \\ U_i(z) &= K_i \frac{zT_s}{z-1} E_i(z) \rightarrow U_i(z) = U_i(z)z^{-1} + K_i z T_s E_i(z) \\ U_d(z) &= K_d \frac{(z-1)}{zT_s} E_d(z) \rightarrow U_d(z) = \frac{K_d}{T_s}\Big(E_d(z) - E_d(z)z^{-1}\Big) \end{aligned} \tag{2-18}$$

with $(E_p(z) = bR(z) - Y(z))$,$(E_i(z) = R(z) - Y(z))$ and $(E_d(z) = cR(z) - Y(z))$. Given the representation of the 2DOF PID controller in the discrete domain showed in Eq. (2-18) the associated difference equation are provided as follows

$$\begin{aligned} u_p(k) &= K_p e_p(k) && with \quad e_p(k) = br(k) - y(k) \\ u_i(k) &= u_i(k-1) + K_i T_s e_i(k) && with \quad e_i(k) = r(k) - y(k) \\ u_d(k) &= \frac{K_d}{T_s}\Big(e_d(k) - e_d(k-1)\Big) && with \quad e_d(k) = cr(k) - y(k) \end{aligned} \tag{2-19}$$

from which it is possible to conclude that the difference equation representing at sample time instant (k) the output of the 2DOF PID controller is

$$u(k) = u_p(k) + u_i(k) + u_d(k) \tag{2-20}$$

It is important to remember that depending of the values of the boolean variables $(enable_{ref_{filter}})$ and $(enable_{der_{filter}})$ Eq. (2-19) and Eq. (2-20) might be modified replacing the reference $(r(k))$ with the filter reference $(r_f(k))$ and the derivative error $(e_d(k))$ with the filtered derivative error $(e_{d_f}(k))$. The equations describing the values of the filtered reference $(r_f(k))$ and of the filtered derivative error term are respectively Eq. (2-8) and Eq. (2-13). However, the 2DOF PID controller output is subjected to saturation constraint which might cause the wind up problem if the integrator term

is present. To avoid this problem an anti wind up back calculation scheme has been developed. Adding the anti wind up integrator term given in Eq. (2-16) to the 2DOF PID controller equation is possible to obtain the final controller output equation.

$$u(k) = u_p(k) + u_i(k) + u_d(k) + u_{aw}(k-1) \tag{2-21}$$

where the anti wind up integrator term ($u_{aw}(k-1)$) is taken at sample $(k-1)$ because it depends on the the saturation error which has not been defined yet when the overall controller output ($u(k)$) is calculated. Therefore the saturation error at sample $(k-1)$ is considered. The saturated overall controller output at time instant (k) is defined as

$$\begin{aligned}
&if\ u(k) \geq U_{Max} \\
&\quad u_{sat}(k) = U_{Max} \\
&else\ if\ u(k) \leq U_{Min} \\
&\quad u_{sat}(k) = U_{Min} \\
&else \\
&\quad u_{sat}(k) = u(k)
\end{aligned}$$

2-1-3 Algorithm

The overall 2DOF PID algorithm has been summarized as follows

Algorithm 1 Two degree of freedom PID controller

Initialization:

Get 2DOF PID controller gain parameters: $(T_s, K_p, K_d, K_i, b, c, U_{Min}, U_{Max})$
Get $(enable_{ref_{filter}}, N_r)$,$(enable_{der_{filter}}, N_d)$,$(enable_{anti_{windup}}, K_{aw})$
Initialization: $(u_{aw_1} = 0,\ u_{i_1} = 0,\ r_{f_1} = 0,\ e_{d_{f_1}} = 0,\ e_{d_1} = 0)$

Algorithm: (Run at frequency equal to $1/T_s$ Hz)

Get reference (ref) and measurement (y)
if $enable_{ref_{filter}} = True$ **then**
    $r_f = (\frac{\frac{1}{N_r}}{\frac{1}{N_r}+T_s})r_{f_1} + (\frac{T_s}{\frac{1}{N_r}+T_s})ref$ (Filtered reference)
    $r = r_f$
else
    $r = ref$
end if
$e_p = br - y$ (Proportional error)
$e_i = r - y$ (Integral error)
$e_d = cr - y$ (Derivative error)
if $enable_{der_{filter}} = True$ **then**
    $e_{d_f} = (\frac{\frac{1}{N_d}}{\frac{1}{N_d}+T_s})e_{d_{f_1}} + (\frac{T_s}{\frac{1}{N_d}+T_s})e_d$ (Filtered derivative error)
    $u_d = \frac{K_d}{T_s}(e_{d_f} - e_{d_{f_1}})$ (Filtered derivative action)
else
    $u_d = \frac{K_d}{T_s}(e_d - e_{d_1})$ (Derivative action)
end if
$u_p = K_p e_p$ (Proportional action)
$u_i = u_{i_1} + K_i T_s e_i$ (Integral action)
$u = u_p + u_i + u_d + u_{aw_1}$ (Controller output not saturated)
if $u \geq U_{Max}$ **then**
    $u_{sat} = U_{Max}$
else if $u \leq U_{Min}$ **then**
    $u_{sat} = U_{Min}$
else
    $u_{sat} = u$
end if
Send u_{sat} (Saturated controller output)
if $enable_{anti_{windup}} = True$ and $K_i \neq 0$ **then**
    $u_{aw} = u_{aw_1} + K_{aw} T_s (u_{sat} - u)$
else
    $u_{aw} = 0$
end if
$u_{aw_1} = u_{aw},\ u_{i_1} = u_i,\ r_{f_1} = r_f,\ e_{d_{f_1}} = e_{d_f},\ e_{d_1} = e_d$ (Update previous values)

2-2 Horizontal position controller module

2-2-1 State space representation

- **Dynamics** :

$$\begin{aligned} \hat{\dot{x}}^W &= \dot{x}^{\star W} \\ \hat{\dot{y}}^W &= \dot{y}^{\star W} \end{aligned} \tag{2-22}$$

- **State space** :
  Given

$$\mathbf{X}_{2\times1} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \hat{x}^W \\ \hat{y}^W \end{bmatrix} \quad \mathbf{Y}_{2\times1} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \hat{x}^W \\ \hat{y}^W \end{bmatrix} \quad \mathbf{U}_{2\times1} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \dot{x}^{\star W} \\ \dot{y}^{\star W} \end{bmatrix} \tag{2-23}$$

  the state space representation of the linear system shown in Eq. (2-22) is

$$\dot{\mathbf{X}}_{2\times1} = A_{2\times2}\mathbf{X}_{2\times1} + B_{2\times2}\mathbf{U}_{2\times1} \tag{2-24}$$

  with

$$A_{2\times2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad B_{2\times2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad C_{2\times2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad D_{2\times2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{2-25}$$

- **Laplace domain** :
  The Laplace transfer function between inputs ($\mathbf{U}(\mathbf{s})$) and outputs ($\mathbf{Y}(\mathbf{s})$) is

$$\mathbf{Y}(\mathbf{s}) = \begin{bmatrix} \frac{1}{s} & 0 \\ 0 & \frac{1}{s} \end{bmatrix} \mathbf{U}(\mathbf{s}) \rightarrow Y_1(s) = \frac{1}{s}U_1(s) \quad Y_2(s) = \frac{1}{s}U_2(s) \tag{2-26}$$

- **Z domain** :
  Applying Backward Euler discretization method $\left(s \approx \frac{z-1}{zT_s}\right)$ Eq. (2-26) becomes

$$\mathbf{Y}(\mathbf{z}) = \begin{bmatrix} \frac{T_s}{1-z^{-1}} & 0 \\ 0 & \frac{T_s}{1-z^{-1}} \end{bmatrix} \mathbf{U}(\mathbf{z}) \rightarrow Y_1(z) = \frac{T_s}{1-z^{-1}}U_1(z) \quad Y_2(z) = \frac{T_s}{1-z^{-1}}U_2(z) \tag{2-27}$$

  where (T_s) represents the chosen sampling time. The difference equations associated to Eq. (2-27) are

$$Y_1(k) = Y_1(k-1) + T_sU_1(k) \quad Y_2(k) = Y_2(k-1) + T_sU_2(k) \tag{2-28}$$

  where (k) represent a generic sample time instant.

2-2-2 Inputs

1. $\underline{\mathbf{T_s^{hpc}}}$: It represents the sampling time of the horizontal position controller module. The latter contains two 2DOF PID controllers running at the same frequency equal to the inverse of the chosen horizontal position controller module sampling time (T_s^{hpc}). The choice of this value is related to the frequency at which the measurements are available. Furthermore having designed the navigation controller framework using a cascade control design it is required to set the horizontal position controller sampling time at least three times lower than the horizontal speed controller module sampling time (T_s^{hsc}). For the sake of this thesis the frequency of the horizontal position controller module has been set equal to $10Hz$. This choice leads to a sampling time equal to ($T_s^{hsc} = 1/10 = 0.1$s).

2. $\underline{\mathbf{par}_{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star} = (\mathbf{K_p}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{K_d}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{K_i}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{b}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{c}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{U_{Min}}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star} \mathbf{U_{Max}}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{enable}_{\mathbf{der_{filter}}}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{N_d}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star} \mathbf{enable}_{\mathbf{ref_{filter}}}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{N_r}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star} \mathbf{enable}_{\mathbf{anti_{windup}}}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star}, \mathbf{K_{aw}}^{(\mathbf{x}^\star,\hat{\mathbf{x}})\rightarrow\dot{\mathbf{x}}^\star})$: They represents the controller tuning parameters associated to the 2DOF PID controller ($2dofPid_{(x^\star,\hat{x})\rightarrow\dot{x}^\star}$). The latter takes as input the error between the desired ($x^{\star W}$) and the estimated ($\hat{x}^W$) position of the quadrotor along the (x) direction expressed in the world coordinate frame. The 2DOF PID controller output represents the desired reference velocity ($\dot{x}^{\star W} \in \begin{bmatrix}-1 & 1\end{bmatrix} \left[m/s\right]$) along the ($x$) direction expressed in world coordinate frame. The choice to bound the velocity value between minus one and one meters per second is related to the small angle approximation assumption allowing to decouple translational quadrotor's motion if the quadrotor movements (pitch and roll angles) are small (15 degree maximum). An illustration describing the 2DOF PID controller together with the discrete plant model used to tune it is provided.

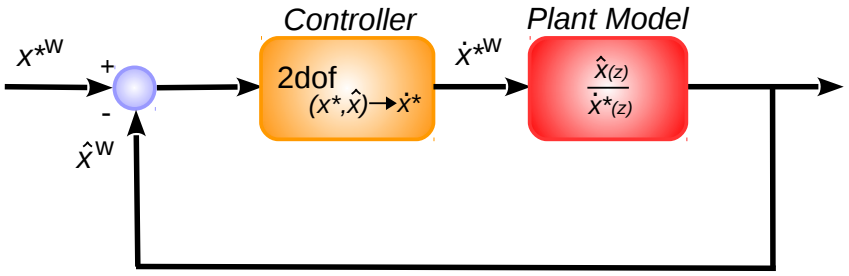


Figure 2-5: This figure shows the input and output of the discrete 2DOF PID controller $(2dofPid_{(x^\star,\hat{x})\to\dot{x}^\star})$ and of the discrete open loop plant model $(\frac{\hat{x}(z)}{\dot{x}^\star(z)})$ used to tune it. The output of the horizontal position controller module is $(\dot{x}^{\star W})$.

The derivation of the discrete plant model used to to tune the $(2dofPid_{(x^\star,\hat{x})\to\dot{x}^\star})$ controller is given in 2-2-1. The discrete model equation is showed in Eq. (2-27). The tuning of the controller parameters have been done ensuring that the controller output is inside the desired boundaries given different types of reference inputs and that the estimated value $(\hat{x}^W)$ will track the reference one $(x^{\star W})$ with no overshoot and a steady state error equal to zero. To be able to see the controller output the $(enable_{der_{filter}})$ boolean variable is set to true and a filter gain (N_d) is chosen such that Eq. (2-10) is satisfied. The choice to introduce a low pass filter on the derivative error term ensures that the control sensitivity function $(\frac{\dot{x}^\star(z)}{x^\star(z)})$ is realizable.

3. $\mathbf{par}_{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star} = (\mathbf{K_p}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{K_d}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{K_i}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{b}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{c}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{U_{Min}}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{U_{Max}}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{enable}_{\mathbf{der_{filter}}}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{N_d}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star} \mathbf{enable}_{\mathbf{ref_{filter}}}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{N_r}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{enable}_{\mathbf{anti_{windup}}}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star}, \mathbf{K_{aw}}^{(\mathbf{y}^\star,\hat{\mathbf{y}})\to\dot{\mathbf{y}}^\star})$: They represent the tuning parameters associated to the 2DOF PID controller $(2dofPid_{(y^\star,\hat{y})\to\dot{y}^\star})$. This controller takes as input the error between the desired $(y^{\star W})$ and the estimated $(\hat{y}^W)$ position along the (y) direction and it provides as output the desired velocity of the quadrotor $(\dot{y}^{\star W} \in \begin{bmatrix}-1 & 1\end{bmatrix} [m/s])$ along the (y) direction . All the data are expressed in world coordinate frame. An illustration of the 2DOF PID controller and of the discrete model used to tune it is provided.

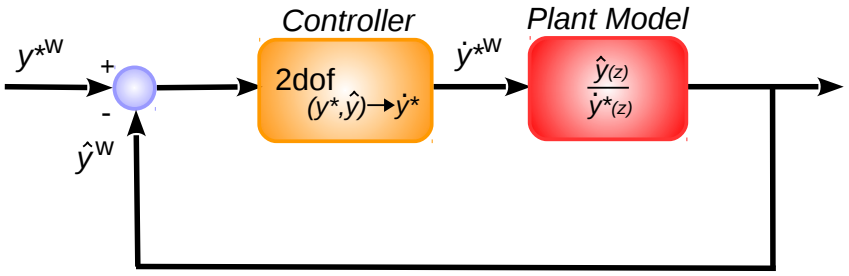


Figure 2-6: This figure shows the input and output of the discrete 2DOF PID ($2dofPid_{(y^\star,\hat{y})\rightarrow\dot{y}^\star}$) controller and of the open loop discrete plant model ($\frac{\hat{y}(z)}{\dot{y}^\star(z)}$) used to tune it. The output of the horizontal position controller module is ($\dot{y}^{\star W}$).

The discrete plant model is given in Eq. (2-27). The derivation of the discrete plant model used to to tune the ($2dofPid_{(y^\star,\hat{y})\rightarrow\dot{y}^\star}$) controller is given in 2-2-1. Also in this case the controller parameters have been tuned such the controller output is inside the boundaries and a steady state error is achieved.

2-2-3 Outputs

1. $\underline{\dot{\mathbf{x}}^{\star\mathbf{W}}, \dot{\mathbf{y}}^{\star\mathbf{W}}}$: They represents the controller outputs provided respectively by the 2DOF PID controller ($2dofPid_{(x^\star,\hat{x})\rightarrow\dot{x}^\star}$) and ($2dofPid_{(y^\star,\hat{y})\rightarrow\dot{y}^\star}$). They are the outputs of the horizontal position controller module and they represents the horizontal quadrotor velocities expressed in world coordinate frame. For the sake of this thesis a saturation constraint has been introduced on both the controller outputs ($\dot{x}^{\star W}, \dot{y}^{\star W} \in \begin{bmatrix}-1 & 1\end{bmatrix} \begin{bmatrix}m/s\end{bmatrix}$). The unit of measure is introduced to underline that although a 2DOF PID controller provides adimensional output it is possible in this case to state that the two controllers outputs values have been tune to have a physical meaning. Indeed they represent velocity.

2-2-4 Algorithm

A summary of the algorithm running inside the horizontal position controller module is provided.

Algorithm 2 Horizontal position controller module

Initialization:

1: Get horizontal position controller module sampling time (T_s^{hpc})

2: Get $2dofPid_{(x^\star,\hat{x})\to\dot{x}^\star}$ tuning parameters:
$par_{(x^\star,\hat{x})\to\dot{x}^\star} = (K_p^{(x^\star,\hat{x})\to\dot{x}^\star}, K_d^{(x^\star,\hat{x})\to\dot{x}^\star}, K_i^{(x^\star,\hat{x})\to\dot{x}^\star}, b^{(x^\star,\hat{x})\to\dot{x}^\star}, c^{(x^\star,\hat{x})\to\dot{x}^\star}, U_{Min}^{(x^\star,\hat{x})\to\dot{x}^\star}, U_{Max}^{(x^\star,\hat{x})\to\dot{x}^\star},$
$enable_{der_{filter}}^{(x^\star,\hat{x})\to\dot{x}^\star}, N_d^{(x^\star,\hat{x})\to\dot{x}^\star}, enable_{ref_{filter}}^{(x^\star,\hat{x})\to\dot{x}^\star}, N_r^{(x^\star,\hat{x})\to\dot{x}^\star}, enable_{anti_{windup}}^{(x^\star,\hat{x})\to\dot{x}^\star}, K_{aw}^{(x^\star,\hat{x})\to\dot{x}^\star})$

3: Get $2dofPid_{(y^\star,\hat{y})\to\dot{y}^\star}$ tuning parameters:
$par_{(y^\star,\hat{y})\to\dot{y}^\star} = (K_p^{(y^\star,\hat{y})\to\dot{y}^\star}, K_d^{(y^\star,\hat{y})\to\dot{y}^\star}, K_i^{(y^\star,\hat{y})\to\dot{y}^\star}, b^{(y^\star,\hat{y})\to\dot{y}^\star}, c^{(y^\star,\hat{y})\to\dot{y}^\star}, U_{Min}^{(y^\star,\hat{y})\to\dot{y}^\star}, U_{Max}^{(y^\star,\hat{y})\to\dot{y}^\star},$
$enable_{der_{filter}}^{(y^\star,\hat{y})\to\dot{y}^\star}, N_d^{(y^\star,\hat{y})\to\dot{y}^\star}, enable_{ref_{filter}}^{(y^\star,\hat{y})\to\dot{y}^\star}, N_r^{(y^\star,\hat{y})\to\dot{y}^\star}, enable_{anti_{windup}}^{(y^\star,\hat{y})\to\dot{y}^\star}, K_{aw}^{(y^\star,\hat{y})\to\dot{y}^\star})$

4: Initialize 2DOF PID controllers:
$2dofPid_{(x^\star,\hat{x})\to\dot{x}^\star} \leftarrow (par_{(x^\star,\hat{x})\to\dot{x}^\star}, T_s^{hpc})$
$2dofPid_{(y^\star,\hat{y})\to\dot{y}^\star} \leftarrow (par_{(y^\star,\hat{y})\to\dot{y}^\star}, T_s^{hpc})$

Algorithm: (Run at frequency equal to $1/T_s^{hpc}$ Hz)

5: Get references: $(x^{\star W}, y^{\star W})$ Unit $\begin{bmatrix} m \end{bmatrix}$

6: Get measurements: $(\hat{x}^W, \hat{y}^W)$ Unit $\begin{bmatrix} m \end{bmatrix}$

7: Calculate $2dofPid_{(x^\star,\hat{x})\to\dot{x}^\star}$ controller output:
$\dot{x}^{\star W} \leftarrow 2dofPid_{(x^\star,\hat{x})\to\dot{x}^\star} \leftarrow (x^{\star W}, \hat{x}^W)$ with $\dot{x}^{\star W} \in \begin{bmatrix} -1 & 1 \end{bmatrix}$ Unit $\begin{bmatrix} m/s \end{bmatrix}$

8: Calculate $2dofPid_{(y^\star,\hat{y})\to\dot{y}^\star}$ controller output:
$\dot{y}^{\star W} \leftarrow 2dofPid_{(y^\star,\hat{y})\to\dot{y}^\star} \leftarrow (y^{\star W}, \hat{y}^W)$ with $\dot{y}^{\star W} \in \begin{bmatrix} -1 & 1 \end{bmatrix}$ Unit $\begin{bmatrix} m/s \end{bmatrix}$

9: Send $(\dot{x}^{\star W}, \dot{y}^{\star W})$

2-3 Vertical position controller module

2-3-1 State space representation

- **Dynamics** :

$$\hat{\dot{z}}^W = \dot{z}^{\star W} \tag{2-29}$$

- **State space** :
Given

$$\mathbf{X} = \hat{z}^W \quad \mathbf{Y} = \hat{z}^W \quad \mathbf{U} = \dot{z}^{\star W} \tag{2-30}$$

the state space representation of the linear system shown in Eq. (2-29) is

$$\dot{\mathbf{X}}_{1\times1} = A_{1\times1}\mathbf{X}_{1\times1} + B_{1\times1}\mathbf{U}_{1\times1} \tag{2-31}$$

with

$$A_{1\times1} = 0 \quad B_{1\times1} = 1 \quad C_{1\times1} = 1 \quad D_{1\times1} = 0 \tag{2-32}$$

- **Laplace domain** :
  The Laplace transfer function between inputs ($\mathbf{U(s)}$) and outputs ($\mathbf{Y(s)}$) is

$$\mathbf{Y(s)} = \frac{1}{s}\mathbf{U(s)} \tag{2-33}$$

- **Z domain** :
  Applying Backward Euler discretization method $\left(s \approx \frac{z-1}{zT_s}\right)$ Eq. (2-33) becomes

$$\mathbf{Y(z)} = \frac{T_s}{1-z^{-1}}\mathbf{U(z)} \tag{2-34}$$

  where (T_s) represents the chosen sampling time. The difference equations associated to Eq. (2-34) are

$$Y(k) = Y(k-1) + T_s U(k) \tag{2-35}$$

  where (k) represent a generic sample time instant.

2-3-2 Inputs

1. $\underline{\mathbf{T_s^{vpc}}}$: It represents the sampling time of the vertical position controller which it is an altitude controller. The choice to separate the horizontal and vertical position controller in two different modules which might have different sampling time it is related to the fact that the altitude measurement can be provided at a higher frequency with respect to the horizontal position measurements. Therefore it is possible to select the frequency of the two modules according to the frequency at which the measurements are available. For the sake of this thesis an EKF is used to provide the quadrotor position, orientation, velocity and acceleration measurements at a frequency close to $30Hz$. Thus, according to the cascade control design it has been chosen to set the vertical position controller module frequency equal to $10Hz$.

2. $\underline{\mathbf{par}_{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star} = (\mathbf{K_p}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{K_d}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{K_i}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{b}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{c}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{U_{Min}}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{U_{Max}}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{enable}_{\mathbf{der_{filter}}}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{N_d}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star} \mathbf{enable}_{\mathbf{ref_{filter}}}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{N_r}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{enable}_{\mathbf{anti_{windup}}}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star}, \mathbf{K_{aw}}^{(\mathbf{z}^\star,\hat{\mathbf{z}})\to\dot{\mathbf{z}}^\star})$: They represent the tuning parameters associated to the 2DOF PID controller ($2dofPid_{(z^\star,\hat{z})\to\dot{z}^\star}$). This controller takes as input the error between the desired ($z^{\star W}$) and the estimated ($\hat{z}^W$) position along the (z) direction and it provides as output the desired velocity of the quadrotor ($\dot{z}^{\star W} \in \begin{bmatrix}-1 & 1\end{bmatrix}\left[m/s\right]$) along the ($z$) direction . All the data are expressed in world coordinate frame. An illustration of the 2DOF PID controller and of the open loop discrete model used to tune it is provided.

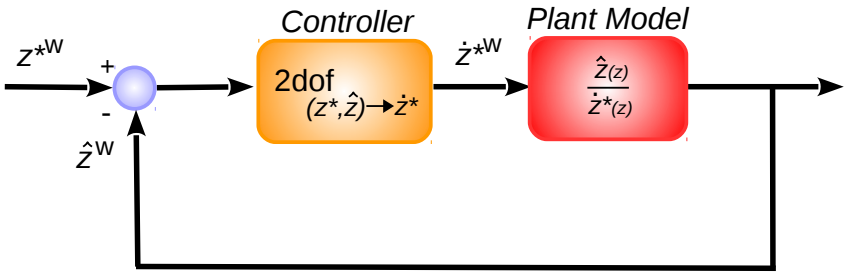


Figure 2-7: This figure shows the input and output of the discrete 2DOF PID $(2dofPid_{(z^\star,\hat{z})\rightarrow\dot{z}^\star})$ controller and of the open loop discrete plant model $(\frac{\hat{z}(z)}{\dot{z}^\star(z)})$ used to tune it. The output of the vertical position controller module is $(\dot{z}^{\star W})$

The discrete plant model is given in Eq. (2-34). The derivation of the discrete open loop plant model used to tune the $(2dofPid_{(z^\star,\hat{z})\rightarrow\dot{z}^\star})$ controller is given in 2-3-1. Also in this case the controller parameters have been tuned such that the controller output is inside the boundaries and a steady state error is achieved.

2-3-3 Outputs

1. $\underline{\dot{\mathbf{z}}^{\star\mathbf{W}}}$: It represents the output of the vertical position controller module. In particular it stands for the desired reference velocity along the (z) direction expressed in world coordinate frame that the quadrotor has to track to ensure that it will reach the desired altitude. For the sake of this thesis the value of this velocity has been bounded between minus one and one meter per second. The ENU convention is used to express the value of $(\dot{z}^{\star W})$ which means that a positive value of the latter will make the quadrotor to move upward whereas a negative value downward. is An illustration of the ENU robot frame together with the navigation framework controller outputs is given in Figure 2-2. A summary of the convention used in the navigation controller framework is given in Table 2-1.

2-3-4 Algorithm

A summary of the algorithm running inside the vertical position controller module is provided.

Algorithm 3 Vertical position controller module

Initialization:

1: Get vertical position controller module sampling time (T_s^{vpc})

2: Get $2dofPid_{(z^\star,\hat{z})\to\dot{z}^\star}$ tuning parameters:
$par_{(z^\star,\hat{z})\to\dot{z}^\star} = (K_p^{(z^\star,\hat{z})\to\dot{z}^\star}, K_d^{(z^\star,\hat{z})\to\dot{z}^\star}, K_i^{(z^\star,\hat{z})\to\dot{z}^\star}, b^{(z^\star,\hat{z})\to\dot{z}^\star}, c^{(z^\star,\hat{z})\to\dot{z}^\star}, U_{Min}^{(z^\star,\hat{z})\to\dot{z}^\star}, U_{Max}^{(z^\star,\hat{z})\to\dot{z}^\star}, enable_{der_{filter}}^{(z^\star,\hat{z})\to\dot{z}^\star}, N_d^{(z^\star,\hat{z})\to\dot{z}^\star}, enable_{ref_{filter}}^{(z^\star,\hat{z})\to\dot{z}^\star}, N_r^{(z^\star,\hat{z})\to\dot{z}^\star}, enable_{anti_{windup}}^{(z^\star,\hat{z})\to\dot{z}^\star}, K_{aw}^{(z^\star,\hat{z})\to\dot{z}^\star})$

3: Initialize 2DOF PID controllers:
$2dofPid_{(z^\star,\hat{z})\to\dot{z}^\star} \leftarrow (par_{(z^\star,\hat{z})\to\dot{z}^\star}, T_s^{vpc})$

Algorithm: (Run at frequency equal to $1/T_s^{vpc}$ Hz)

4: Get references: $(z^{\star W})$ Unit $\begin{bmatrix} m \end{bmatrix}$

5: Get measurements: $(\hat{z}^W)$ Unit $\begin{bmatrix} m \end{bmatrix}$

6: Calculate $2dofPid_{(z^\star,\hat{z})\to\dot{z}^\star}$ controller output:
$\dot{z}^{\star W} \leftarrow 2dofPid_{(z^\star,\hat{z})\to\dot{z}^\star} \leftarrow (z^{\star W}, \hat{z}^W)$ with $\dot{z}^{\star W} \in \begin{bmatrix} -1 & 1 \end{bmatrix}$ Unit $\begin{bmatrix} m/s \end{bmatrix}$

7: Send $(\dot{z}^{\star W})$

2-4 Horizontal speed controller module

2-4-1 State space representation

- **Dynamics** :

$$\begin{aligned} \hat{\ddot{x}}^W &= \frac{T^{\star W \approx R}}{m} \begin{bmatrix} cos(\hat{\psi}^W) & sin(\hat{\psi}^W) \end{bmatrix} \begin{bmatrix} \theta^{\star R} \\ \phi^{\star R} \end{bmatrix} \\ \hat{\ddot{y}}^W &= \frac{T^{\star W \approx R}}{m} \begin{bmatrix} sin(\hat{\psi}^W) & -cos(\hat{\psi}^W) \end{bmatrix} \begin{bmatrix} \theta^{\star R} \\ \phi^{\star R} \end{bmatrix} \end{aligned} \tag{2-36}$$

Linearizing around the equilibrium point ($T^{\star W \approx R} = mg$) Eq. (2-36) becomes

$$\begin{aligned} \hat{\ddot{x}}^W &= \frac{1}{g} \begin{bmatrix} cos(\hat{\psi}^W) & sin(\hat{\psi}^W) \end{bmatrix} \begin{bmatrix} \theta^{\star R} \\ \phi^{\star R} \end{bmatrix} \\ \hat{\ddot{y}}^W &= \frac{1}{g} \begin{bmatrix} sin(\hat{\psi}^W) & -cos(\hat{\psi}^W) \end{bmatrix} \begin{bmatrix} \theta^{\star R} \\ \phi^{\star R} \end{bmatrix} \end{aligned} \tag{2-37}$$

- **State space** :

Given

$$\begin{aligned}\mathbf{X}_{2\times1} &= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \hat{x}^W \\ \hat{y}^W \end{bmatrix} \quad \mathbf{Y}_{2\times1} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \hat{x}^W \\ \hat{y}^W \end{bmatrix} \\ \mathbf{U}_{2\times1} &= \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} cos(\hat{\psi}^W) & sin(\hat{\psi}^W) \\ sin(\hat{\psi}^W) & -cos(\hat{\psi}^W) \end{bmatrix} \begin{bmatrix} \theta^{\star R} \\ \phi^{\star R} \end{bmatrix} \\ \begin{bmatrix} \theta^{\star R} \\ \phi^{\star R} \end{bmatrix} &= \begin{bmatrix} cos(\hat{\psi}^W) & sin(\hat{\psi}^W) \\ sin(\hat{\psi}^W) & -cos(\hat{\psi}^W) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}\end{aligned} \tag{2-38}$$

the state space representation of the linear system shown in Eq. (2-36) is

$$\dot{\mathbf{X}}_{2\times1} = A_{2\times2}\mathbf{X}_{2\times1} + B_{2\times2}\mathbf{U}_{2\times1} \tag{2-39}$$

with

$$A_{2\times2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad B_{2\times2} = \begin{bmatrix} \frac{1}{g} & 0 \\ 0 & \frac{1}{g} \end{bmatrix} \quad C_{2\times2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad D_{2\times2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{2-40}$$

- **Laplace domain** :
  The Laplace transfer function between inputs ($\mathbf{U(s)}$) and outputs ($\mathbf{Y(s)}$) is

$$\mathbf{Y(s)} = \begin{bmatrix} \frac{g}{s} & 0 \\ 0 & \frac{g}{s} \end{bmatrix} \mathbf{U(s)} \rightarrow Y_1(s) = \frac{g}{s}U_1(s) \quad Y_2(s) = \frac{g}{s}U_2(s) \tag{2-41}$$

- **Z domain** :
  Applying Backward Euler discretization method $\left(s \approx \frac{z-1}{zT_s}\right)$ Eq. (2-41) becomes

$$\mathbf{Y(z)} = \begin{bmatrix} \frac{gT_s}{1-z^{-1}} & 0 \\ 0 & \frac{gT_s}{1-z^{-1}} \end{bmatrix} \mathbf{U(z)} \rightarrow Y_1(z) = \frac{gT_s}{1-z^{-1}}U_1(z) \quad Y_2(z) = \frac{gT_s}{1-z^{-1}}U_2(z) \tag{2-42}$$

  where (T_s) represents the chosen sampling time and (g) is the gravity force value. The difference equations associated to Eq. (2-42) are

$$Y_1(k) = Y_1(k-1) + gT_sU_1(k) \quad Y_2(k) = Y_2(k-1) + gT_sU_2(k) \tag{2-43}$$

  where (k) represent a generic sample time instant.

2-4-2 Inputs

1. $\underline{\mathbf{T_s^{vsc}}}$: It represents the sampling time value chosen for the horizontal speed controller controller module. The latter contains two 2DOF PID controller in charge of making the quadrotor able to track an horizontal desired velocity. Because the EKF used in this thesis provides data at $30Hz$ it has been chosen to set the horizontal speed controller module sampling time equal to ($T_s^{hsc} = 1/30 = 0.033$s).

2. $\mathbf{par}_{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}} = (\mathbf{K_p}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{K_d}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{K_i}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{b}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{c}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{U_{Min}}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{U_{Max}}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{enable}_{\mathbf{der_{filter}}}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{N_d}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}} \mathbf{enable}_{\mathbf{ref_{filter}}}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{N_r}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{enable}_{\mathbf{anti_{windup}}}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}}, \mathbf{K_{aw}}^{(\dot{\mathbf{x}}^\star,\hat{\dot{\mathbf{x}}})\rightarrow\theta_\mathbf{v}})$: They represent the tuning parameters associated to the 2DOF PID controller $(2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v})$. This controller takes as input the error between the desired $(\dot{x}^{\star W})$ and the estimated $(\hat{\dot{x}}^W)$ velocity along the (x) direction and it provides as output the pitch that the quadrotor has to track expressed in world coordinate frame (θ_v) with $(\theta_v \in \begin{bmatrix}-0.2 & 0.2\end{bmatrix}\begin{bmatrix}rad\end{bmatrix})$. However, this value does not represent the pitch of the quadrotor expressed in robot coordinate frame required to be sent to the Parrot and Pixhawk autopilot. For this reason a transformation taking into account the quadrotor's yaw $(\hat{\psi}^W)$ and the roll angle (ϕ_v, output of $(2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v})$) expressed in world coordinate frame are required. The explanation of how the transformation is derived in given in Appendix B-4 and the transformation equation is showed in Eq. (2-38). Thus it is possible to conclude that the value of the pitch angle expressed in robot coordinate frame is given by

$$\theta^{\star R} = cos(\hat{\psi}^W)\theta_v + sin(\hat{\psi}^W)\phi_v \tag{2-44}$$

Choosing to set both the controller outputs $(\theta_v, \phi_v \in \begin{bmatrix}-0.2 & 0.2\end{bmatrix}\begin{bmatrix}rad\end{bmatrix})$ and knowing that $(\psi^W \in \begin{bmatrix}-\pi & \pi\end{bmatrix}\begin{bmatrix}rad\end{bmatrix})$ it is possible to state that

$$\theta^{\star R} \in \begin{bmatrix}-\sqrt{-(\theta_{v_{Min}}^2 + \phi_{v_{Min}}^2)} & \sqrt{(\theta_{v_{Max}}^2 + \phi_{v_{Max}}^2)}\end{bmatrix} = \begin{bmatrix}-0.2828 & 0.2828\end{bmatrix}\begin{bmatrix}rad\end{bmatrix} \tag{2-45}$$

which are equivalent to $(\pm 16.2°)$. An illustration of the 2DOF PID controller and of the open loop discrete model used to tune it is provided.

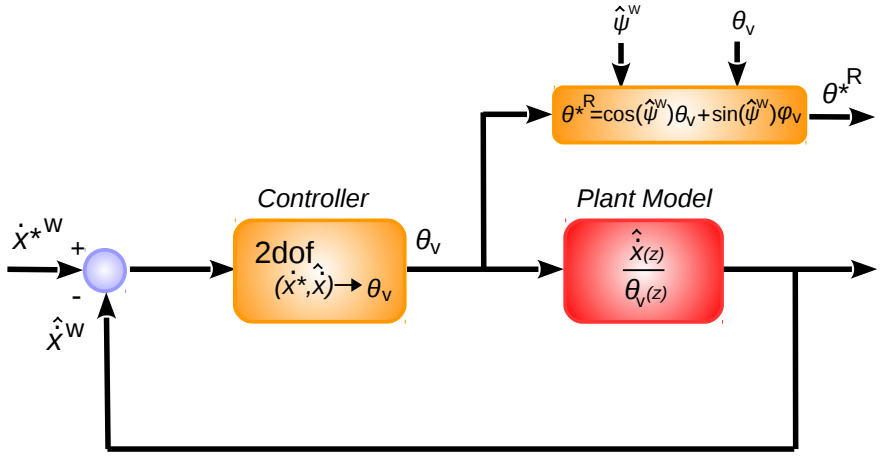


Figure 2-8: This figure shows the input and output of the discrete 2DOF PID $(2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v})$ controller and of the open loop discrete plant model $(\frac{\hat{\dot{x}}(z)}{\theta_v(z)})$ used to tune it. It also shows how the controller output (θ_v) is used in combination with the estimated yaw angle $(\hat{\psi})$ and the $(2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v})$ controller output (ϕ_v) to derive the output of the horizontal speed controller module $(\theta^{\star R})$.

The discrete plant model is given in Eq. (2-42). The derivation of the discrete open loop plant model used to tune the $(2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v})$ controller is given in 2-4-1. The tuning of this controller has been done ensuring that the controller output is inside the boundaries and a steady state error is achieved.

3. $\mathbf{par}_{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}} = (\mathbf{K_p}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{K_d}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{K_i}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{b}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{c}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{U_{Min}}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{U_{Max}}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{enable}_{\mathbf{der_{filter}}}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{N_d}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}} \mathbf{enable}_{\mathbf{ref_{filter}}}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{N_r}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{enable}_{\mathbf{anti_{windup}}}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}}, \mathbf{K_{aw}}^{(\dot{\mathbf{y}}^\star,\hat{\dot{\mathbf{y}}})\rightarrow\phi_\mathbf{v}})$: They represent the tuning parameters associated to the 2DOF PID controller $(2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v})$. This controller takes as input the error between the desired $(\dot{y}^{\star W})$ and the estimated $(\hat{\dot{y}}^W)$ velocity along the (y) direction and it provides as output the roll angle that the quadrotor has to track expressed in world coordinate frame (ϕ_v) with $(\phi_v \in \begin{bmatrix}-0.2 & 0.2\end{bmatrix}\ [rad])$. To be able to be sent to the Parrot or Pixhawk autopilot the roll angle has to be mapped from world coordinate to robot coordinate frame. To this the quadrotor's yaw $(\hat{\psi}^W)$ and the pitch angle $(\theta_v$, output of $(2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v}))$ are required. A detail explanation of how the mapping from world to robot frame is achieved is given in Appendix B-4 and the matrix used to map the roll and pitch from world to robot frame coordinate is presented in Eq. (2-38).

Thus it is possible to conclude that the value of the desired roll angle ($\phi^{\star R}$) expressed in robot coordinate frame is given by

$$\phi^{\star R} = sin(\hat{\psi}^W)\theta_v - cos(\hat{\psi}^W)\phi_v \tag{2-46}$$

Setting the controller outputs saturation values to ($\theta_v, \phi_v \in \begin{bmatrix}-0.2 & 0.2\end{bmatrix}\begin{bmatrix}rad\end{bmatrix}$) it is possible to calculate knowing that ($\psi^W \in \begin{bmatrix}-\pi & \pi\end{bmatrix}\begin{bmatrix}rad\end{bmatrix}$) the saturation boundaries associated to the roll angle expressed in robot coordinate frame.

$$\phi^{\star R} \in \begin{bmatrix}-\sqrt{-(\theta_{v_{Min}}^2 + \phi_{v_{Min}}^2)} & \sqrt{(\theta_{v_{Max}}^2 + \phi_{v_{Max}}^2)}\end{bmatrix} = \begin{bmatrix}-0.2828 & 0.2828\end{bmatrix}\begin{bmatrix}rad\end{bmatrix} \tag{2-47}$$

which are equivalent to ($\pm 16.2°$). An illustration of the 2DOF PID controller and of the open loop discrete model used to tune it is provided.

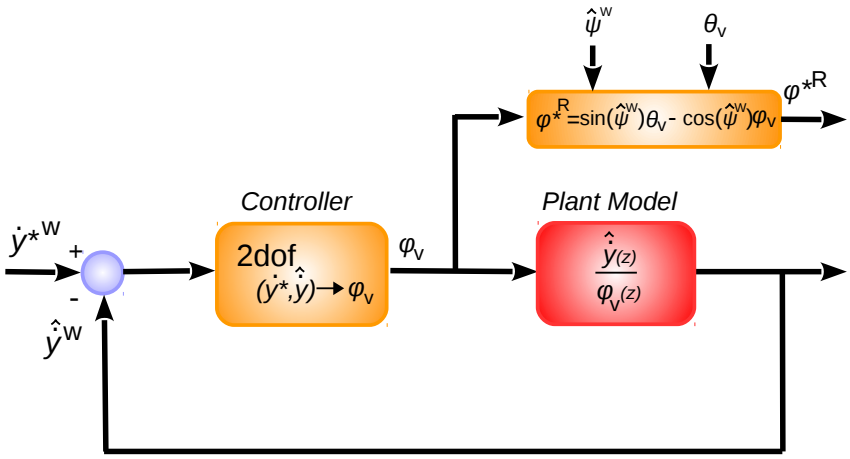


Figure 2-9: This figure shows the input and output of the discrete 2DOF PID ($2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v}$) controller and of the open loop discrete plant model ($\frac{\hat{\dot{y}}(z)}{\phi_v(z)}$) used to tune it. It also shows how the controller output (ϕ_v) is used in combination with the estimated yaw angle ($\hat{\psi}$) and the ($2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v}$) controller output ($\theta_v$) to derive the output of the horizontal speed controller module ($\phi^{\star R}$).

The discrete plant model is given in Eq. (2-42). The derivation of the discrete open loop plant model used to tune the ($2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v}$) controller is given in 2-4-1. The tuning of this controller has been done ensuring that the controller output ($\phi^{\star R}$) is inside the boundaries and a steady state error is achieved.

4. $\underline{\hat{\psi}^{\mathbf{W}}}$: It represents the estimated yaw angle of the quadrotor expressed in world coordinate frame. Usually this value is provided by the EKF but it can also be obtained using a MOCAP system. To be precise it represents how much (degree or radians) the quadrotor's robot frame is rotated with respect to the world coordinate frame only considering the robot frame rotation about the (z) axis of the world coordinate frame.

2-4-3 Outputs

1. $\underline{\theta^{\star\mathbf{R}}, \phi^{\star\mathbf{R}}}$: They are respectively the desired pitch and roll angles expressed in robot coordinate frame that it has been chosen to send to the quadrotor's autopilot. To calculate the two values two different 2DOF PID controller has been used ($2dofPid_{(\dot{x}^{\star},\hat{\dot{x}})\rightarrow\theta_v}$) and ($2dofPid_{(\dot{y}^{\star},\hat{\dot{y}})\rightarrow\phi_v}$). It has been chosen to limit these values at around ($\pm 16°$) to ensure that the small angle approximation assumption is satisfied. The ENU convention is used to represents the two outputs value. This choice leads to say that a positive value of ($\theta^{\star R}$) makes to quadrotor to move forward whereas a negative value backward. On the other hand a positive value of ($\phi^{\star R}$) makes the quadrotor to move right whereas a negative value to move left. An illustration of the ENU robot frame together with the navigation framework controller outputs is given in Figure 2-2. A summary of the convention used in the navigation controller framework is given in Table 2-1.

2-4-4 Algorithm

A summary of the algorithm running inside the horizontal speed controller module is provided.

Algorithm 4 Horizontal speed controller module

Initialization:

1: Get horizontal speed controller module sampling time (T_s^{hsc})

2: Get $2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}$ tuning parameters:
$par_{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} = (K_p^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, K_d^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, K_i^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, b^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, c^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, U_{Min}^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, U_{Max}^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v},$
$enable_{der_{filter}}^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, N_d^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, enable_{ref_{filter}}^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, N_r^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, enable_{anti_{windup}}^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, K_{aw}^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v})$

3: Get $2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}$ tuning parameters:
$par_{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} = (K_p^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, K_d^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, K_i^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, b^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, c^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, U_{Min}^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, U_{Max}^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v},$
$enable_{der_{filter}}^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, N_d^{(\dot{y}^\star,\hat{\dot{x}})\to\phi_v}, enable_{ref_{filter}}^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, N_r^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, enable_{anti_{windup}}^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, K_{aw}^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v})$

4: Initialize 2DOF PID controllers:
$2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} \leftarrow (par_{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}, T_s^{hsc})$
$2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} \leftarrow (par_{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}, T_s^{hsc})$

Algorithm: (Run at frequency equal to $1/T_s^{hsc}$ Hz)

5: Get yaw angle measurement: $(\hat{\psi}^W) \in \begin{bmatrix} -\pi & \pi \end{bmatrix}$ Unit $\begin{bmatrix} rad \end{bmatrix}$

6: Get references: $(\dot{x}^{\star W}, \dot{y}^{\star W}) \in \begin{bmatrix} -1 & 1 \end{bmatrix}$ Unit $\begin{bmatrix} m/s \end{bmatrix}$

7: Get measurements: $(\hat{\dot{x}}^W, \hat{\dot{y}}^W)$ Unit $\begin{bmatrix} m/s \end{bmatrix}$

8: Calculate $2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v}$ controller output:
$\theta_v \leftarrow 2dofPid_{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} \leftarrow (\dot{x}^{\star W}, \hat{\dot{x}}^W)$ with $\theta_v \in \begin{bmatrix} -0.2 & 0.2 \end{bmatrix}$ Unit $\begin{bmatrix} rad \end{bmatrix}$

9: Calculate $2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v}$ controller output:
$\phi_v \leftarrow 2dofPid_{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} \leftarrow (\dot{y}^{\star W}, \hat{\dot{y}}^W)$ with $\phi_v \in \begin{bmatrix} -0.2 & 0.2 \end{bmatrix}$ Unit $\begin{bmatrix} rad \end{bmatrix}$

10: Calculate reference pitch: $\theta^{\star R} \leftarrow cos(\hat{\psi}^W)\theta_v + sin(\hat{\psi}^W)\phi_v$

11: Calculate reference roll: $\phi^{\star R} \leftarrow sin(\hat{\psi}^W)\theta_v - cos(\hat{\psi}^W)\phi_v$

12: Send $(\theta^{\star R}, \phi^{\star R})$ with $\theta^{\star R}, \phi^{\star R} \in \begin{bmatrix} -0.2828 & 0.2828 \end{bmatrix}$ Unit $\begin{bmatrix} rad \end{bmatrix}$

2-5 Vertical speed controller module

2-5-1 State space representation

- **Dynamics** :

$$\hat{\ddot{z}}^W = \frac{T^{\star W \approx R}}{m} - g \tag{2-48}$$

- **State space** :
Given

$$\begin{aligned} &\mathbf{X} = \hat{\dot{z}}^W \quad \mathbf{Y} = \hat{\dot{z}}^W \quad \mathbf{U} = \frac{T^{\star W \approx R}}{m} - g \\ &T^{\star W \approx R} = mU + g \end{aligned} \tag{2-49}$$

the state space representation of the linear system shown in Eq. (2-48) is

$$\dot{\mathbf{X}}_{1\times 1} = A_{1\times 1}\mathbf{X}_{1\times 1} + B_{1\times 1}\mathbf{U}_{1\times 1} \tag{2-50}$$

with

$$A_{1\times1} = 0 \quad B_{1\times1} = 1 \quad C_{1\times1} = 1 \quad D_{1\times1} = 0 \tag{2-51}$$

- **Laplace domain** :
  The Laplace transfer function between inputs ($\mathbf{U(s)}$) and outputs ($\mathbf{Y(s)}$) is

$$\mathbf{Y(s)} = \frac{1}{s}\mathbf{U(s)} \tag{2-52}$$

- **Z domain** :
  Applying Backward Euler discretization method $\left(s \approx \frac{z-1}{zT_s}\right)$ Eq. (2-52) becomes

$$\mathbf{Y(z)} = \frac{T_s}{1-z^{-1}}\mathbf{U(z)} \tag{2-53}$$

  where (T_s) represents the chosen sampling time. The difference equations associated to Eq. (2-53) are

$$Y(k) = Y(k-1) + T_s U(k) \tag{2-54}$$

  where (k) represent a generic sample time instant.

2-5-2 Inputs

1. $\underline{\mathbf{T_s^{vsc}}}$: It represents the sampling time of the vertical speed controller module. Inside this module a 2DOF PID controller in charge of calculating the desired quadrotor thrust given the desired and estimation velocity along the (z) direction has been developed. This controller module is used only in dealing with quadrotor equipped with Pixhawk autopilot. In dealing with Parrot quadrotors it is not required the use of this module because the autopilot accepts the desired velocity along the (z) direction as input which it is the output of the vertical position controller. The sampling time chosen for this module is ($T_s^{vsp} = 1/30 = 0.033s$). This sampling time choice is related to the fact that the EKF used to get the estimated velocity along the (z) direction provides data at ($30Hz$).

2. $\underline{\mathbf{scale_{thrust}}}$: In dealing with the Pixhawk autopilot it is required to move the quadrotor to send a thrust command value between zero and one. This means that the 2DOF PID controller has to generate an output in this range. However, in this case the output has lost the physical meaning because it does not represent a force (Newton). However, it is still possible to tune the controller to ensure that the controller output is bounded between zero and one. Furthermore, to avoid that a too large thrust is applied to the motor the ($scale_{thrust}$) variable has been introduced. The latter has the aim to prevent that large thrust value are applied to the quadrotor. This variable is really useful to understand what is the quadrotor minimum thrust to overcome gravity. Indeed in using the Pixhawk autopilot quadrotors with different motors the minimum thrust to overcome gravity will change. To avoid to send a too large thrust which might cause the quadrotor to crash it is useful to start increasing slowly the ($scale_{thrust}$) variable up the point that the quadrotor starts moving overcoming gravity. After having found the minimum required thrust value to overcome gravity it is possible to use the ($scale_{thrust}$)

variable to select the desired maximum thrust value. If a different quadrotor with smaller size and less powerful motors is used it sufficient to decrease the $(scale_{thrust})$ value without modifying the controller parameters to make the quadrotor fly. To summarize the output of the vertical speed controller module is obtained multiplying the output of 2DOF PID controller (T_v) with the $(scale_{thrust})$ factor.

$$T^{\star R} = T_v scale_{thrust} \quad with \quad T_v, scale_{thrust} \in \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{2-55}$$

3. $\mathbf{par}_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}} = (\mathbf{K_p}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{K_d}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{K_i}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{b}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{c}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{U_{Min}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{U_{Max}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{enable}_{\mathbf{der_{filter}}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{N_d}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}} \mathbf{enable}_{\mathbf{ref_{filter}}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{N_r}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{enable}_{\mathbf{anti_{windup}}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}}, \mathbf{K_{aw}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow \mathbf{T_v}})$: They represent the controller tuning parameters associated to the 2DOF PID controller $(2dofPid_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v})$. This controller takes as input the error between the desired $(\dot{z}^{\star W})$ and the estimated $(\hat{\dot{z}}^W)$ velocity and it provides as output a thrust value $(T_v \in \begin{bmatrix} 0 & 1 \end{bmatrix})$. The output of the vertical speed controller module is the thrust value (T_v) provided by the 2DOF PID controller multiplied times the $(scale_{thrust})$ factor as showed in Eq. (2-55). An illustration of the 2DOF PID controller and of the open loop discrete model used to tune it is provided.

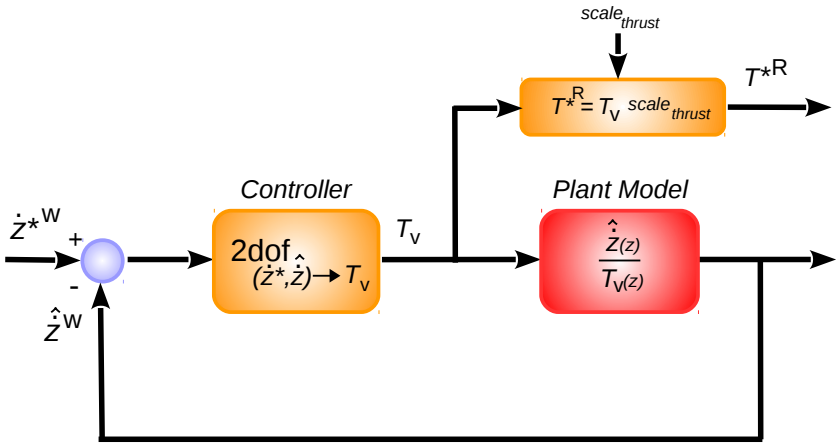


Figure 2-10: This figure shows the input and output of the discrete 2DOF PID $(2dofPid_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v})$ controller and of the open loop discrete plant model $(\frac{\hat{\dot{z}}(z)}{T_v(z)})$ used to tune it. It also shows how the controller output (T_v) is used in combination with the $(scale_{thrust})$ value to derived the output of the vertical speed controller module $(T^{\star R \approx W})$.

The discrete plant model is given in Eq. (2-53). The derivation of the discrete open loop plant model used to tune the ($2dofPid_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}$) controller is given in 2-5-1. The tuning of this controller has been done ensuring that the controller output is inside the boundaries and a steady state error is achieved.

2-5-3 Outputs

1. ($\mathbf{T^{\star R\approx W}}$) : It represents the output of the vertical speed controller module and it is calculated according to Eq. (2-55). In dealing with the Pixhawk autopilot this output is bounded between zero and one. To be precise it represents how much force is applied by the motor to make the quadrotor move. The upper script ($R \approx W$) is used to indicate that the (z) direction of the robot frame is close the (z) direction of the world frame. This occurs because small angle approximation assumption is satisfied. For this reason it is possible to say that the quadrotor thrust can be referred to both robot and world coordinate frame. The ENU convention is used to describe the thrust value ($T^{\star R\approx W}$). Because the latter is mapped between zero and one is possible to say that increasing the thrust value the quadrotor will move upward where decreasing it the quadrotor will move downward. An illustration of the ENU robot frame together with the navigation framework controller outputs is given in Figure 2-2. A summary of the convention used in the navigation controller framework is given in Table 2-1.

2-5-4 Algorithm

A summary of the algorithm running inside the vertical speed controller module is provided.

Algorithm 5 Vertical speed controller module

Initialization:

1: Get vertical speed controller module sampling time (T_s^{vsc})
2: Get thrust scale value: $scale_{thrust} \in \begin{bmatrix} 0 & 1 \end{bmatrix}$
3: Get $2dofPid_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}$ tuning parameters:
$par_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v} = (K_p^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, K_d^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, K_i^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, b^{\hat{\dot{z}}\rightarrow \dot{z}^\star}, c^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, U_{Min}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, U_{Max}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v},$
$enable_{der_{filter}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, N_d^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, enable_{ref_{filter}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, N_r^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, enable_{anti_{windup}}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, K_{aw}^{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v})$
4: Initialize 2DOF PID controllers:
$2dofPid_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v} \leftarrow (par_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}, T_s^{vsc})$

Algorithm: (Run at frequency equal to $1/T_s^{vsc}$ Hz)

5: Get references: $(\dot{z}^{\star W}) \in \begin{bmatrix} -1 & 1 \end{bmatrix}$ Unit $\begin{bmatrix} m/s \end{bmatrix}$
6: Get measurements: $(\hat{\dot{z}}^W)$ Unit $\begin{bmatrix} m/s \end{bmatrix}$
7: Calculate $2dofPid_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v}$ controller output:
$T_v \leftarrow 2dofPid_{(\dot{z}^\star,\hat{\dot{z}})\rightarrow T_v} \leftarrow (\dot{z}^{\star W}, \hat{\dot{z}}^W)$ with $T_v \in \begin{bmatrix} 0 & 1 \end{bmatrix}$ (Pixhawk thrust boundaries)
8: $T^{*W\approx R} = T_v scale_{thrust}$
9: Send ($T^{*W\approx R}$)

2-6 Yaw controller module

2-6-1 State space representation

- **Dynamics** :

$$\dot{\hat{\psi}}^W = \dot{\psi}^{\star W \approx R} \tag{2-56}$$

- **State space** :
  Given

$$\mathbf{X} = \hat{\psi}^W \quad \mathbf{Y} = \hat{\psi}^W \quad \mathbf{U} = \dot{\psi}^{\star W \approx R} \tag{2-57}$$

  the state space representation of the linear system shown in Eq. (2-56) is

$$\dot{\mathbf{X}}_{1\times1} = A_{1\times1}\mathbf{X}_{1\times1} + B_{1\times1}\mathbf{U}_{1\times1} \tag{2-58}$$

  with

$$A_{1\times1} = 0 \quad B_{1\times1} = 1 \quad C_{1\times1} = 1 \quad D_{1\times1} = 0 \tag{2-59}$$

- **Laplace domain** :
  The Laplace transfer function between inputs ($\mathbf{U}(\mathbf{s})$) and outputs ($\mathbf{Y}(\mathbf{s})$) is

$$\mathbf{Y}(\mathbf{s}) = \frac{1}{s}\mathbf{U}(\mathbf{s}) \tag{2-60}$$

- **Z domain** :
  Applying Backward Euler discretization method $\left(s \approx \frac{z-1}{zT_s}\right)$ Eq. (2-60) becomes

$$\mathbf{Y}(\mathbf{z}) = \frac{T_s}{1-z^{-1}}\mathbf{U}(\mathbf{z}) \tag{2-61}$$

  where (T_s) represents the chosen sampling time. The difference equations associated to Eq. (2-61) are

$$Y(k) = Y(k-1) + T_s U(k) \tag{2-62}$$

  where (k) represent a generic sample time instant.

2-6-2 Inputs

1. $\underline{\mathbf{T_s^{\psi c}}}$: It represents the sampling time of the yaw controller module. The choice of this sampling time depends on the yaw angle measurement of the robot frame with respect to the world coordinate frame. In this thesis it has been chosen to set ($T_s^{\psi c} = 1/10 = 0.1s$). The yaw measurement is available also at a higher frequency however it is convenient to avoid to control the yaw angle at a higher frequency rate to ensure that the assumption stating that the rotational and translation quadrotor's motion are decoupled is satisfied.

2. $\mathbf{par}_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = (\mathbf{K_p}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{K_d}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{K_i}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{b}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{c}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{U_{Min}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} \mathbf{U_{Max}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{enable}_{\mathbf{der_{filter}}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{N_d}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} \mathbf{enable}_{\mathbf{ref_{filter}}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{N_r}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} \mathbf{enable}_{\mathbf{anti_{windup}}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, \mathbf{K_{aw}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star})$: They represents the controller tuning parameters associated to the 2DOF PID controller ($2dofPid_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}$). This controller takes as input the error between the desired ($\psi^{\star W} \in \begin{bmatrix}-\pi & \pi\end{bmatrix}\begin{bmatrix}rad\end{bmatrix}$) and the estimated ($\hat{\psi}^W \in \begin{bmatrix}-\pi & \pi\end{bmatrix}\begin{bmatrix}rad\end{bmatrix}$) yaw angle and it provides as output an angular velocity value ($\psi^{\star R \approx W} \in \begin{bmatrix}-0.4 & 0.4\end{bmatrix}\begin{bmatrix}rad/s\end{bmatrix}$). To control the yaw angle ensuring that the quadrotor will always rotate along the direction in which the yaw error is smaller it is required to modify the calculated yaw error (e_v) as following

$$\begin{aligned} &if\ e_v > \pi \\ &\quad e_\psi = e_v - 2\pi \\ &else\ if\ e_v < -\pi \\ &\quad e_\psi = e_v + 2\pi \\ &else \\ &\quad e_\psi = e_v \end{aligned} \tag{2-63}$$

where ($e_v = \psi^{\star W} - \hat{\psi}^W$) represents the error between the desired and the measured yaw angle. Thus it is possible to conclude that the input error of the 2DOF PID ($2dofPid_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}$) controller is ($e_\psi$). An illustration of the 2DOF PID controller and of the open loop discrete model used to tune it is provided.

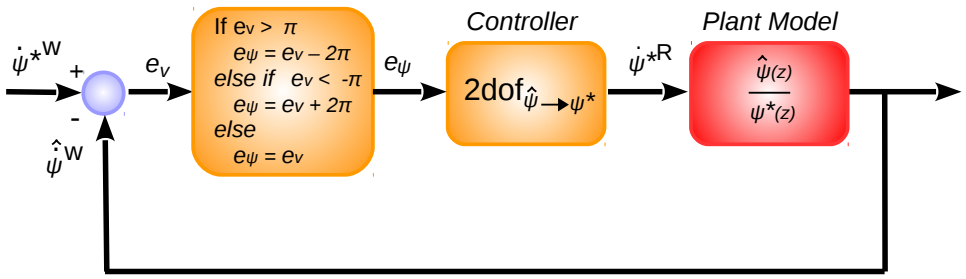


Figure 2-11: This figure shows the input and output of the discrete 2DOF PID $(2dofPid_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star})$ controller and of the open loop discrete plant model $(\frac{\hat{\psi}(z)}{\dot{\psi}^\star(z)})$ used to tune it. Furthermore, it also shows how to modify the yaw error to ensure that the quadrotor will always rotate clockwise or counterclockwise depending on the direction in which the yaw error is smaller. The controller output of the yaw controller module is $(\dot{\psi}^{\star R\approx W})$.

The discrete plant model is given in Eq. (2-61). The derivation of the discrete open loop plant model used to tune the $(2dofPid_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star})$ controller is given in 2-6-1. The tuning of this controller has been done ensuring that the controller output is inside the boundaries and a steady state error is achieved.

2-6-3 Outputs

1. $(\dot{\psi}^{\star\mathbf{R}\approx\mathbf{W}})$: It is the output of the yaw controller module and it represents the yaw angular rate that the quadrotor has to track to make itself reach the desired yaw angle. This value follow ENU convention which means that if the yaw rate is positive the quadrotor will rotate counterclockwise whereas if it is negative it will rotate clockwise. An illustration of the ENU robot frame together with the navigation framework controller outputs is given in Figure 2-2. A summary of the convention used in the navigation controller framework is given in Table 2-1.

2-6-4 Algorithm

A summary of the algorithm running inside the yaw controller module is provided.

Algorithm 6 Yaw controller module

Initialization:

Get yaw controller module sampling time ($T_s^{\psi c}$)

Get $2dofPid_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}$ tuning parameters:

$par_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = (K_p^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, K_d^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, K_i^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, b^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, c^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, U_{Min}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, U_{Max}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, enable_{der_{filter}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, N_d^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, enable_{ref_{filter}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, N_r^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, enable_{anti_{windup}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, K_{aw}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star})$

Initialize 2DOF PID controllers:

$2dofPid_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} \leftarrow (par_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}, T_s^{\psi c})$

Algorithm: (Run at frequency equal to $1/T_s^{\psi c}$ Hz)

Get references: $(\psi^{\star W}) \in \begin{bmatrix} -\pi & \pi \end{bmatrix}$ Unit $\begin{bmatrix} rad \end{bmatrix}$

Get measurements: $(\hat{\psi}^W) \in \begin{bmatrix} -\pi & \pi \end{bmatrix}$ Unit $\begin{bmatrix} rad \end{bmatrix}$

$e_\psi = \psi^{\star W} - \hat{\psi}^W$

if $e_\psi > \pi$ **then**

  $r_\psi = e_\psi - 2\pi$

  $y_\psi = 0$

else if $e_\psi < -\pi$ **then**

  $r_\psi = e_\psi + 2\pi$

  $y_\psi = 0$

else

  $r_\psi = \psi^{\star W}$

  $y_\psi = \hat{\psi}^W$

end if

Calculate $2dofPid_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star}$ controller output:

$\dot{\psi}^{\star R\approx W} \leftarrow 2dofPid_{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} \leftarrow (r_\psi, y_\psi)$ with $\dot{\psi}^{\star R\approx W} \in \begin{bmatrix} -0.4 & 0.4 \end{bmatrix}$ Unit $\begin{bmatrix} rad/s \end{bmatrix}$

Send $(\dot{\psi}^{\star R\approx W})$

Chapter 3

Vision based planner to approach either a static or moving object

After having completed the design of the navigation controller framework the goal of this chapter is to present a vision based planner composed by a perception, state estimator and image based visual servo module able to generate the desired reference velocities required to drive the quadrotor up to a certain desired distance from the chosen visual marker minimizing the error between the estimated visual marker center position on the image plane and the desired one. An illustration of the problem is showed in the bottom figure.

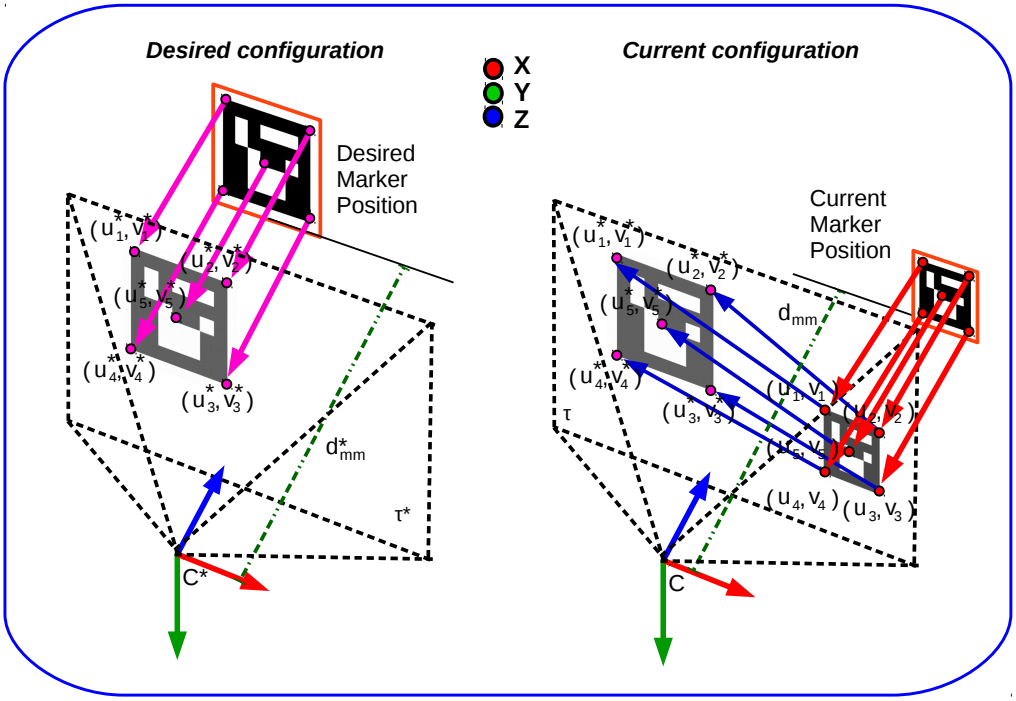


Figure 3-1: The figure shows on the left side the desired pixel values (pink) computed knowing the desired center location $(u_5^\star, v_5^\star)$ of the visual marker on the image plane $(\tau^\star)$ and the desired distance between visual marker and camera $(d_{mm}^\star)$. On the right side is shown the image based visual servo control problem which consist in driving the detected pixel points (red) towards the desired ones (pink). The aim of the IBVS controller is to move the camera frame $(\mathbf{C})$ attached with the quadrotor itself such that the detected pixel points (red) appearing on the current camera image plane (τ) will move towards the desired ones (pink).

Figure 3-1 shows the vision based planner problem which is solved combining three different modules.

- **Perception Module**: This module is in charge of detecting the chosen visual marker and extract the pixel points (red) $(u_{i=1,2\cdots5} = row, v_{i=1,2\cdots5} = column)$ representing the markers' corners and the center. Furthermore, it also generates the desired pixel points (pink) given both the desired center position of the visual marker on the image plane plane $(\tau^\star)$ and the desired distance between visual marker and chosen quadrotor camera.
- **State Estimator Module**: This module takes as input the detected pixel points (red) and it provides an estimation of them that is used when a detection is lost for a small amount of time.
- **Image Based Visual Servo Module**: This module is in charge given both the desired and the estimated pixel points of computing the desired velocities that the quadrotor has track to drive asymptotically the error between the desired and the detected pixel points to zero.

An illustration of the vision based planner architecture combined with the navigation controller framework is provided.

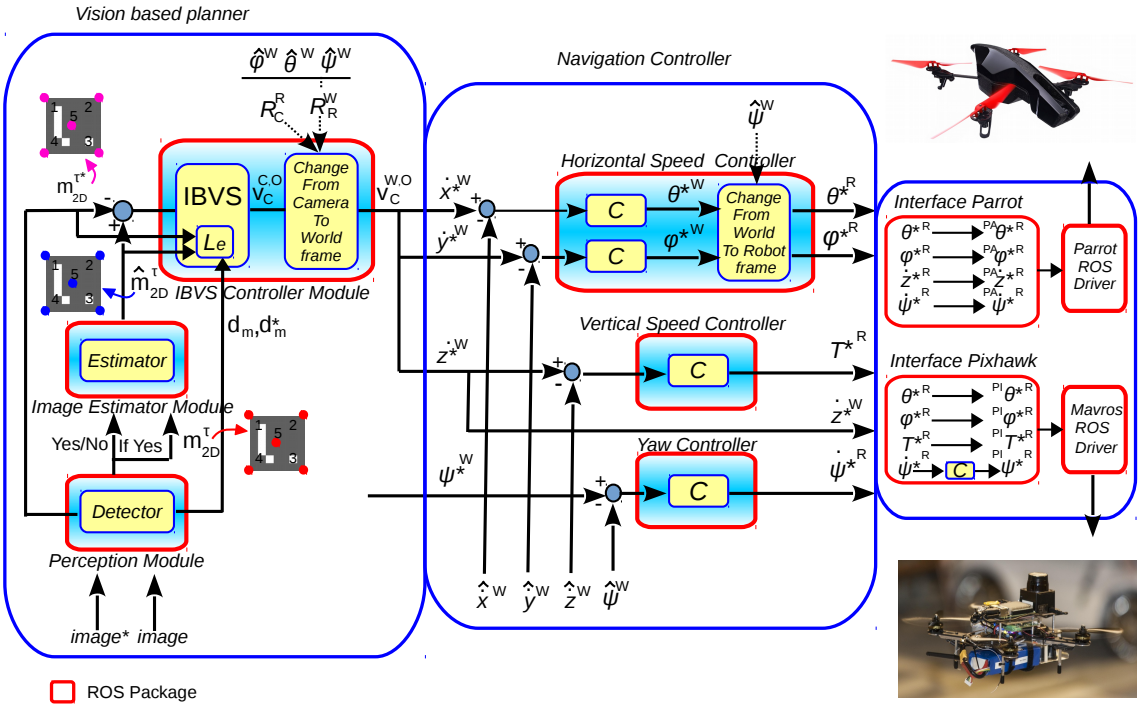


Figure 3-2: Vision based planner combined with navigation controller framework

In dealing with the vision based planner only the horizontal and vertical speed controllers are required because the image based visual servo works in the velocity domain. Therefore the horizontal and vertical position controllers have been removed by the navigation controller framework because there are not required to accomplish the vision based planner task. The yaw controller appears in the navigation controller framework because it is possible to control independently the yaw angle while the vision based planner is active. In controlling the yaw is important to avoid that a quadrotor rotation will make the visual marker disappear from the acquired chosen camera image.

3-1 Perception module

Th goal of the perception module is the extraction of the center and of the four corners of the chosen visual marker in pixel (row,column) and $(2D)$ image coordinates. The detector is the key element of this module and it is in charge of extracting the pixel coordinate values of the center of the visual marker and the size of its side in pixel coordinates unit. For the sake of this thesis the ArUco detector from the ArUco library [5] has been chosen and the ArUco

markers have been used to solve the vision based planner task. The choice to use ArUco as a visual marker is mainly due to the fact that the ArUco detector available in the ArUco library works at a frequency of around 30 Hz which allows to extract the center of the chosen marker and its side in pixel coordinates unit almost every time an image is acquired. For a first trial of the algorithm it has been preferred to rely on a good detector to exploit firstly how the vision based planner performs with high frequency detection.

3-1-1 Inputs

1. **chosen_camera** : It is an integer value indicating from which camera (0=bottom) or (1=front) the quadrotor has to acquire the image to solve the vision based planner task. A quadrotor is usually equipped with a front and a bottom camera from which images can be acquired at a specific rate usually around 30Hz. However the perception module does not know from which camera it has to get the image. Furthermore, the vision based planner should be designed to be able to be used either with the front or the bottom quadrotor's camera image. For example using the front quadrotor's camera the vision based planner can be used to follow a visual marker placed on an moving object or simply approach to it, if the latter is static, to release a certain item or perform some manipulation (if a robotic arm is provided). On the other hand the vision based planner used in combination with the image acquired by the bottom quadrotor's camera can be used to land either on a moving or on a static platform where the visual marker has been previously located. For this reason in designing the perception module an integer variable called (*chosen_camera*) has been introduced. The latter if it is set to zero it will ensure that the perception module will get both the bottom camera image and the intrinsic matrix and distortion coefficients associated to it. On the other hand if the variable is set to one the front camera image and the intrinsic and distortion coefficients associated to the front camera will be used. The drawback of this approach is that both the two cameras are required to be active at the same time and this might increase the computational cost of the algorithm. The vision based planner will choose which image has to use according to the (*chosen_camera*) variable which is set as default to one (front camera images used) but that can be changed online either by the mission planner or by the user itself. When the perception module is initialized if no (*chosen_camera*) value is provided the (*chosen_camera*) values is set to one by default. Dealing with two cameras there is the need to load two different set of configuration variables associated to the front and bottom camera. This occurs because in a mission the quadrotor has to be able to use the vision based planner with both front and bottom camera without the need to switch off the quadrotor and load new parameters. For example, the vision based planner can be used with the front camera to approach a specific ArUco marker with (${}^{f}ID$) number equals to (${}^{f}ID^{\star} = 5$) at a desired distance (${}^{f}d^{\star}_{mm} = 1000$) (millimeters) and later with the bottom camera to approach a different ArUco marker with different side dimension (${}^{b}side^{\star}_{mm} = 170$) (millimeters) and different (${}^{b}ID^{\star} = 9$) at a different desired distance (${}^{b}d^{\star}_{mm} = 500$) (millimeters) from it. Therefore, it is required that the perception module is able to switch easily between front and bottom camera. The upper scripts (f) and (b) are used to differentiate between default fixed variables associated respectively to the front and to the bottom camera.
2. $({}^{\mathbf{f}}\mathbf{u}^{\star}_{\mathbf{5}}, {}^{\mathbf{f}}\mathbf{v}^{\star}_{\mathbf{5}}, {}^{\mathbf{b}}\mathbf{u}^{\star}_{\mathbf{5}}, {}^{\mathbf{b}}\mathbf{v}^{\star}_{\mathbf{5}})$: They represent respectively the desired ($u = row, v = column$)

pixel coordinates values of the center of the ArUco marker respectively in the image acquired by the front and by the bottom camera. Thus given an image acquired by the chosen camera a desired ideal marker will appear on the acquired image with $(row, column)$ pixel coordinate value equal to $(u_5^\star, v_5^\star)$. It is important to be able to choose freely the location of the desired center on the acquired image to avoid to have the latter always located on the center of the acquired image itself. An example where it is convenient to set the desired center of the ArUco marker in a different location from the image center is a mission where the quadrotor has to release an object inside a bucket. Assuming that the object to release is located under the quadrotor, having the desired center of the ArUco marker in the center of the acquired bottom image will require to place the ArUco marker on top of the bucket. However, this choice will prevent the object to be released to fall inside the bucket itself. For this reason having the possibility to locate the desired center of the visual marker in a different location from the image's center will make the quadrotor able to release the object inside the bucket simply setting the desired center to the left or to the right of the bucket itself. Also in this case different desired center location can be selected for the front and bottom camera according to the task that the quadrotor has to do. It is important to avoid to place the desired center of the visual marker close to the border of the image. If this occurs the image based visual servo control law will not perform nicely because it is really simple that given a quadrotor movement the ArUco marker will disappear from the image. The goal of the defaults parameters $({}^f u_5^\star, {}^f v_5^\star)$ and $({}^b u_5^\star, {}^b v_5^\star)$ is to accurately select the desired center of the visual marker on the acquired chosen image. The perception module will choose to load either the desired center of the front or bottom camera according to the $(chosen\_camera)$ value (0=bottom, 1 = front).

3. $(\mathbf{^f ID}^\star, \mathbf{^b ID}^\star)$: They represent the (ID) numbers of the desired ArUco that the detector inside the perception module has to detect according to the chosen camera value (front or bottom). It is like having two different markers like an helipad and a circle associated to the front and bottom camera. The choice to work with ArUco visual markers allows to have different visual markers available for different tasks which are identified by a unique (ID) number. An illustration showing different ArUco markers is provided.

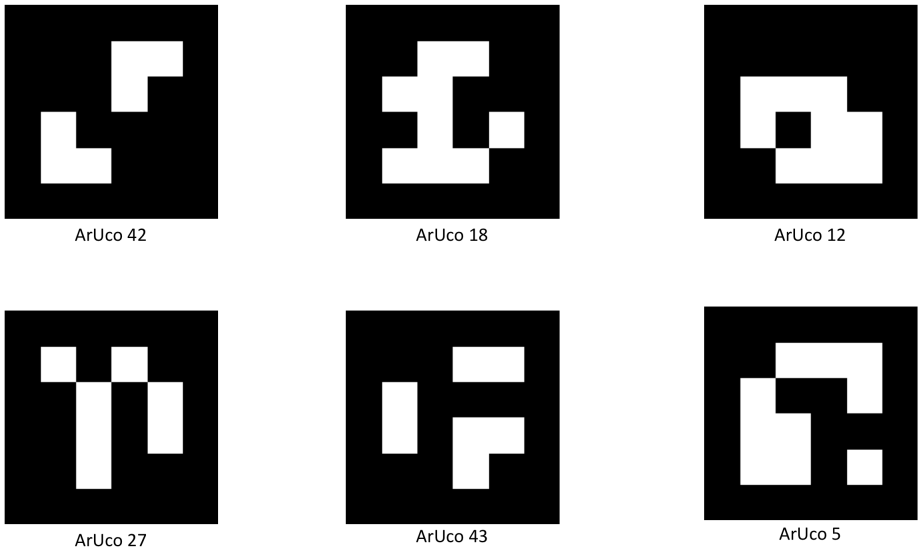


Figure 3-3: ArUco markers

The same detector is used to detect all the ArUco markers showed in Figure 3-3 and it provides as a output the center, the side dimension in pixel and the (ID) of the detected marker. For example, if the front camera has been chosen ($chosen\_camera = 1$) and the ArUco marker appearing in the front image has the same (ID) number of $({}^{f}ID^{\star})$, the corners and the center of the detected marker will be extracted otherwise not. Having the possibility to associate different markers to front and bottom camera images will allow to place different ArUco markers in the same environment without the need to ensure that when for example the landing task is started only an ArUco marker is present in the scenario. In addition visual markers such as ArUco markers can be used also to improve the estimation of the position of the quadrotor with respect to the chosen fixed world coordinate frame if the position of the marker with respect to the world is known a priori. This means that it is possible that in a scenario different ArUco markers will appear.

4. $({}^{\mathbf{f}}\mathbf{d}^{\star}_{\mathbf{mm}}, {}^{\mathbf{b}}\mathbf{d}^{\star}_{\mathbf{mm}})$: They are the desired distance expressed in millimeters between either the front or the bottom camera and the chosen ArUco marker with desired $(ID^{\star})$ number equal either to $({}^{f}ID^{\star})$ or $({}^{b}ID^{\star})$. The desired distance between chosen camera and chosen ArUco markers is used to calculate the desired pixel corners location knowing the desired center of the marker $(u_5^{\star}, v_5^{\star})$. Furthermore, in the image based visual servo module the desired distance $(d^{\star}_{mm})$ can also be used to compute the desired interaction matrix that is required to derive the desired velocities required by the quadrotor to approach the visual marker at the desired distance $(d^{\star}_{mm})$ minimizing the error between the desired center $(u_5^{\star}, v_5^{\star})$ and the detected one (u_5, v_5). In selecting the desired distance between chosen camera and visual marker is important to ensure that at the chosen desired distance the chosen camera is able to detect the marker. Setting the desired distance to zero for example does not make any sense because when the camera gets to close to the marker the latter disappear from the image itself. The same consideration can be done in setting a to large desired distance between camera and visual marker.

Thus the desired distance is in between the minimum distance between camera and visual marker such that a detection occurs and the maximum distance at which it is possible to detect the marker. This minimum and maximum distance value depend on the chosen camera, on the size of the visual marker and on the chosen detector.

5. $(\underline{{}^{\mathbf{f}}\mathbf{side}_{\mathbf{mm}}, {}^{\mathbf{b}}\mathbf{side}_{\mathbf{mm}}})$: They represent the dimension in millimeters of the side of the chosen ArUco markers identified by $({}^{f}ID^{\star})$ or $({}^{b}ID^{\star})$. According to which camera is enable a specific ArUco marker is chosen and it is side is measured and it is used to both estimate the distance between the chosen camera and the marker and to calculate the desired marker's corners.

6. $(\underline{{}^{\mathbf{f}}\mathbf{f}_{\mathbf{mm}}, {}^{\mathbf{f}}\mathbf{s}_{\mathbf{h_{mm}}}, {}^{\mathbf{f}}\mathbf{im}_{\mathbf{h_{pix}}}, {}^{\mathbf{b}}\mathbf{f}_{\mathbf{mm}}, {}^{\mathbf{b}}\mathbf{s}_{\mathbf{h_{mm}}}, {}^{\mathbf{b}}\mathbf{im}_{\mathbf{h_{pix}}}})$: They are default variables associated to the chosen camera. Usually the values of these variables are found looking at the technical camera specification. To be precise $(f_{mm}, s_{h_{mm}}, im_{h_{pix}})$ stand for respectively the absolute focal length (millimeters), the camera sensor height (millimeters) and the number of rows of the chosen image (front or bottom). A quadrotor can have different front and bottom cameras and for this reason different values are required to differentiate between front and bottom default camera parameters.

7. $(\underline{\mathbf{R}^{\mathbf{R}}_{\mathbf{C_f}}, \mathbf{R}^{\mathbf{R}}_{\mathbf{C_b}}})$: These two rotation matrices stand for how either the front or bottom camera frame is rotated with respect to the robot coordinate frame. They are fixed rotation matrices because usually the front and the bottom quadrotor cameras are fixed with the quadrotor frame. The robot frame follow ENU convention and a pinhole camera model is used to model both the front and bottom camera. An illustration is provided to show world, robot, front and bottom camera frame.

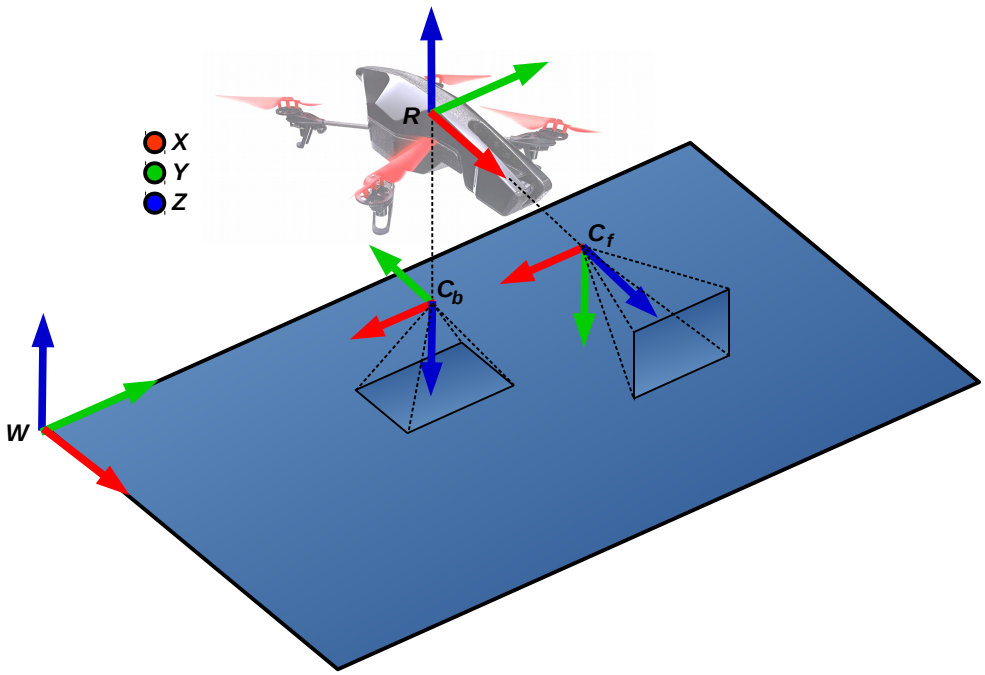


Figure 3-4: Illustration of World coordinate frame ($\mathbf{W}$), Robot coordinate frame ($\mathbf{R}$), Front camera coordinate frame ($\mathbf{C_f}$) and Bottom camera coordinate frame ($\mathbf{C_b}$). For both front and bottom camera frame a pinhole camera model has been chosen. The World and Robot coordinate frame follow ENU convention.

Given Figure 3-4 is possible to calculate the matrices $(R^R_{C_f}, R^R_{C_b})$ as following

$$R^R_{C_f} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad and \quad R^R_{C_b} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3-1}$$

3-1-2 Outputs

1. $(\mathbf{m^{\tau^\star}_{pix_{i=1,2,\cdots 5}}}, \mathbf{m^{\tau^\star}_{pix_{i=1,2,\cdots 5}}})$: They represent respectively the desired pixel $(u_i^\star, v_i^\star) = (row_i, column_i)$ and $(2D)$ image coordinates $(x^\star_{2D_i}, y^\star_{2D_i})$ of the corners $(i = 1, 2, 3, 4)$ and of the center $(i = 5)$ of the chosen marker.
According to the *chosen_camera* value the front $({}^f u_5^\star,\ {}^f v_5^\star,\ {}^f d^\star_{mm},\ {}^f side_{mm},\ {}^f ID^\star,\ {}^f f_{mm},\ {}^f s_{h_{mm}},\ {}^f im_{h_{pix}},\ R^R_{C_f}\ )$ or the bottom $({}^b u_5^\star,\ {}^b v_5^\star,\ {}^b d^\star_{mm},\ {}^b side_{mm},\ {}^b ID^\star,\ {}^b f_{mm},\ {}^b s_{h_{mm}},\ {}^b im_{h_{pix}},\ R^R_{C_b}\ )$ default camera parameters are loaded into the variables $(u_5^\star,\ v_5^\star,\ d^\star_{mm},\ side_{mm},\ ID^\star,\ f_{mm},\ s_{h_{mm}},\ im_{h_{pix}},\ R^R_C\ )$. To compute $(m^{\tau^\star}_{pix_{i=1,2,\cdots 5}}, m^{\tau^\star}_{pix_{i=1,2,\cdots 5}})$ the variables $(u_5^\star,\ v_5^\star,\ d^\star_{mm},\ side_{mm},\ f_{mm},\ s_{h_{mm}},\ im_{h_{pix}}\ )$ are used to derive the top left, top right, bottom right, bottom left corners of the visual marker with respect to the desired center $(u_5^\star, v_5^\star)$. Assuming that the marker is planar the side of the desired

marker expressed in pixel is given by

$$side^{\star}_{pix} = \frac{f_{mm} \;\; side_{mm} \;\; im_{h_{pix}}}{d^{\star}_{mm} \;\; s_{h_{mm}}} \tag{3-2}$$

The derivation of Eq. (3-2) is given in Appendix C-1. Given the side of the desired marker in pixel ($side^{\star}_{pix}$) and the desired marker center location $(u^{\star}_5, v^{\star}_5)$ the desired row and column pixel coordinate values of the marker's corners are given by

$$\begin{aligned}
(u^{\star}_1, v^{\star}_1) &= (\frac{im_{h_{pix}}}{2} - \frac{side^{\star}_{pix}}{2}, \frac{im_{w_{pix}}}{2} - \frac{side^{\star}_{pix}}{2}) \quad \text{top left} \\
(u^{\star}_2, v^{\star}_2) &= (\frac{im_{h_{pix}}}{2} - \frac{side^{\star}_{pix}}{2}, \frac{im_{w_{pix}}}{2} + \frac{side^{\star}_{pix}}{2}) \quad \text{top right} \\
(u^{\star}_3, v^{\star}_3) &= (\frac{im_{h_{pix}}}{2} + \frac{side^{\star}_{pix}}{2}, \frac{im_{w_{pix}}}{2} + \frac{side^{\star}_{pix}}{2}) \quad \text{bottom right} \\
(u^{\star}_4, v^{\star}_4) &= (\frac{im_{h_{pix}}}{2} + \frac{side^{\star}_{pix}}{2}, \frac{im_{w_{pix}}}{2} - \frac{side^{\star}_{pix}}{2}) \quad \text{bottom left}
\end{aligned} \tag{3-3}$$

where $(u_{i=1,2,\cdots 4}, v_{i=1,2\cdots 4})$ are respectively the row and the column pixel values. To summarize the desired pixel coordinate values are

$$m^{\tau^{\star}}_{pix_{i=1,2,\cdots 5}} = \begin{bmatrix} {u_1}^{\star} \\ {v_1}^{\star} \\ {u_2}^{\star} \\ {v_2}^{\star} \\ \vdots \\ {u_5}^{\star} \\ {v_5}^{\star} \end{bmatrix} \tag{3-4}$$

After having calculated the desired pixel coordinate values given the intrinsic camera parameters (f_x, f_y, c_x, c_y) associated to the chosen camera and derived from a previous offline camera calibration (using $Matlab$ [6] or $OpenCV$ [7]) it is possible to calculate the $(2D)$ image coordinate associated to $(m^{\tau^{\star}}_{pix_{i=1,2,\cdots 5}})$. Using perspective projection [8] the desired $(2D)$ image coordinates $(m^{\tau^{\star}}_{2D_{i=1,2\cdots 5}})$ are

$$m^{\tau^{\star}}_{2D_{i=1,2\cdots 5}} = \begin{bmatrix} x^{\star}_{2D_1} \\ y^{\star}_{2D_1} \\ x^{\star}_{2D_2} \\ y^{\star}_{2D_2} \\ \vdots \\ x^{\star}_{2D_5} \\ y^{\star}_{2D_5} \end{bmatrix} = \begin{bmatrix} \frac{v^{\star}_1 - c_x}{f_x} \\ \frac{u^{\star}_1 - c_y}{f_y} \\ \frac{v^{\star}_2 - c_x}{f_x} \\ \frac{u^{\star}_2 - c_y}{f_y} \\ \vdots \\ \frac{v^{\star}_5 - c_x}{f_x} \\ \frac{u^{\star}_5 - c_y}{f_y} \end{bmatrix} \tag{3-5}$$

A scheme summarizing how to calculate $(m^{\tau^\star}_{pix_{i=1,2\cdots5}}, m^{\tau^\star}_{2D_{i=1,2\cdots5}})$ is provided for the sake of clarity.

$$\begin{bmatrix} {}^f u_5^\star, {}^f v_5^\star, {}^f d^\star_{mm}, {}^f side_{mm} \\ {}^f f_{mm}, {}^f s_{h_{mm}}, {}^f im_{h_{pix}} \\ {}^f f_x, {}^f f_y, {}^f c_x, {}^f c_y \end{bmatrix} \longrightarrow \begin{bmatrix} u_5^* \\ v_5^* \\ d^*_{mm} \\ side_{mm} \\ f_{mm} \\ s_{h_{mm}} \\ im_{h_{pix}} \\ f_x \\ f_y \\ c_x \\ c_y \end{bmatrix} \rightarrow \begin{bmatrix} m^{\tau^\star}_{pix_{i=1,2,\cdots5}} \\ m^{\tau^\star}_{2D_{i=1,2\cdots5}} \end{bmatrix}$$

$$chosen\_camera = 0 \; or \; 1$$

$$\begin{bmatrix} {}^b u_5^\star, {}^b v_5^\star, {}^b d^\star_{mm}, {}^b side_{mm} \\ {}^b f_{mm}, {}^b s_{h_{mm}}, {}^b im_{h_{pix}} \\ {}^b f_x, {}^b f_y, {}^b c_x, {}^b c_y \end{bmatrix} \longrightarrow$$

Another option to compute the desired references is to take a picture of the visual marker where the latter will appear in the acquired image in the desired position. Given this desired image it is possible to extract the desired corners and center using the detector. It has been chosen to make the computation of the desired references $(m^{\tau^\star}_{pix_{i=1,2,\cdots5}}, m^{\tau^\star}_{2D_{i=1,2\cdots5}})$ independent from the chosen visual marker to avoid to have different desired images given different markers and to allow to place the desired center precisely on the image plane. An illustration of the extracted corners given different desired centers location but same desired distance between visual marker and camera is provided in the bottom figure.

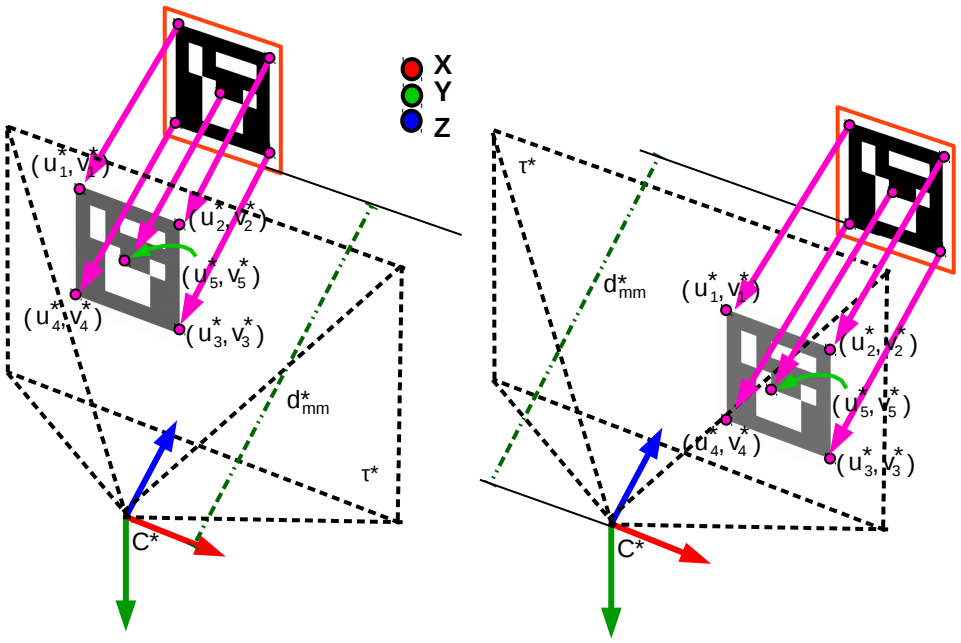


Figure 3-5: The figure shows how given different desired center pixel coordinate values $(u_5^\star, v_5^\star)$ and a desired distance $(d_{mm}^\star)$ in millimeters between camera and visual marker is possible to located the desired corners (pink) of the chosen ArUco marker freely on the desired image plane $(\tau^\star)$.

2. $\underline{(\mathbf{m}^\tau_{\mathbf{pix_{i=1,2,\cdots 5}}}, \mathbf{m}^\tau_{\mathbf{pix_{i=1,2,\cdots 5}}})}$: They stand for respectively the detected pixel $(u_i, v_i) = (row_i, column_i)$ and $(2D)$ image coordinates (x_{2D_i}, y_{2D_i}) of the corners $(i = 1, 2, 3, 4)$ and of the center $(i = 5)$ of the chosen visual marker. To calculate $(m^\tau_{pix_{i=1,2,\cdots 5}}, m^\tau_{2D_{i=1,2\cdots 5}})$ a detector is required able to extract the corners and the center of the detected marker. Using the ArUco marker detector of the ArUco library is possible given a certain image where the ArUco marker appears to extract the ArUco marker's corners, the center, the perimeter and also the ArUco marker (ID) number. The detector orders the ArUco corners in a clockwise direction from the top left corner according to the front side of the ArUco. However if the ArUco rotate the detected corners will rotate as well with the marker. A figure showing how the ArUco detector outputs the corners given different ArUco marker orientation is provided.

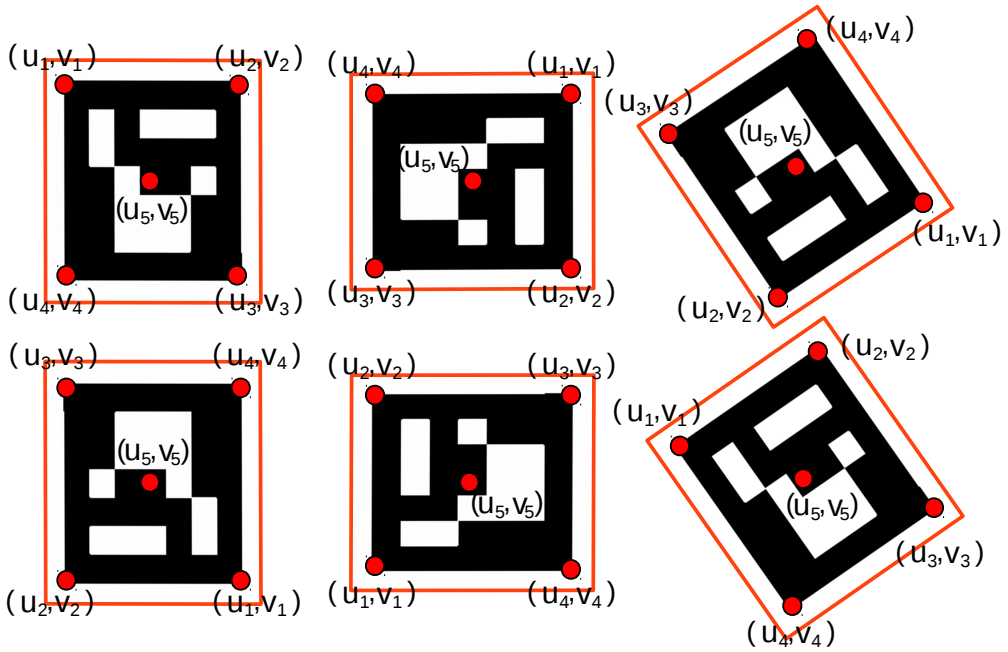


Figure 3-6: This figure shows the output of the ArUco detector (red points) when the ArUco marker has different orientation.

The output showed in Figure 3-6 is not optimal in dealing with an under actuated system such as the quadrotor. Indeed, the latter cannot rotate itself without translating which means that it is impossible for the quadrotor to ensure that the desired marker's corners and center will coincide with the detected ones if a pitch or roll rotation on the marker is applied. A solution to overcome this problem is to create the detected corners knowing the side and the center of the detected ArUco marker at every iteration. In this way the side of the ArUco will change depending on the detected side but the orientation of the detected corners is independent from the ArUco marker orientation. An illustration showing how using the side and the center of the ArUco marker the detected corners become independent from the ArUco marker orientation is provided.

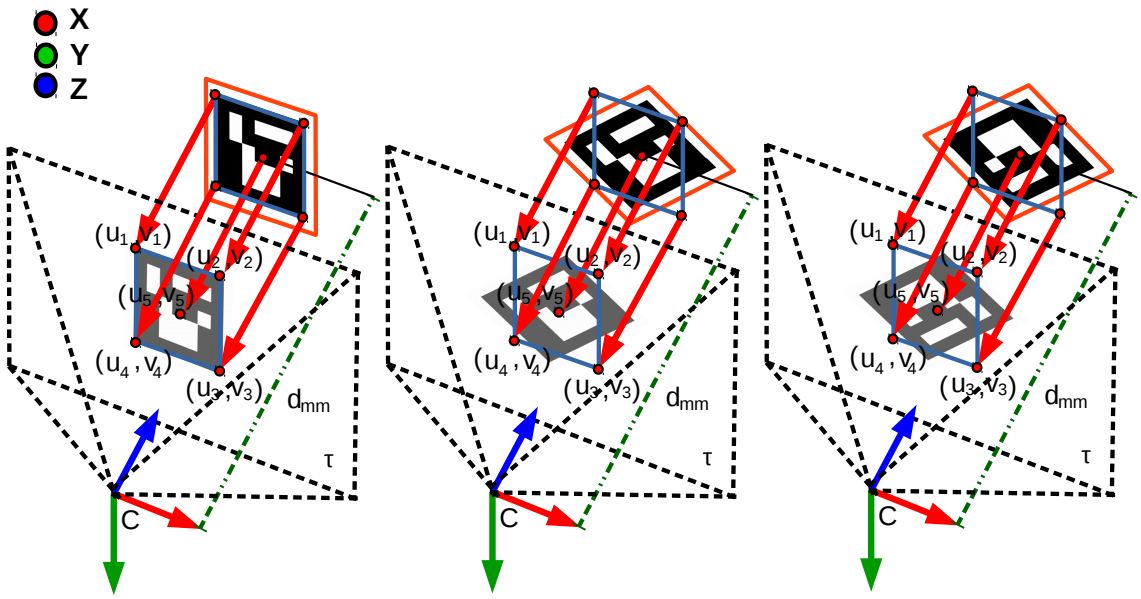


Figure 3-7: The figure shows how the detected corners (red) are independent from the ArUco marker orientation. The latter only depends on the side and on the center of the detected marker.

Given the detected side of the ArUco ($side_{pix}$) obtained divided the perimeter by four and the detected center of it (u_5, v_5) the corners of the ArUco marker are derived as

$$\begin{aligned}
(u_1, v_1) &= (u_5 - \frac{side_{pix}}{2}, v_5 - \frac{side_{pix}}{2}) \quad \text{top left} \\
(u_2, v_2) &= (u_5 - \frac{side_{pix}}{2}, v_5 + \frac{side_{pix}}{2}) \quad \text{top right} \\
(u_3, v_3) &= (u_5 + \frac{side_{pix}}{2}, v_5 + \frac{side_{pix}}{2}) \quad \text{bottom right} \\
(u_4, v_4) &= (u_5 + \frac{side_{pix}}{2}, v_5 - \frac{side_{pix}}{2}) \quad \text{bottom left}
\end{aligned} \tag{3-6}$$

a matrix representation is provided

$$m^{\tau}_{pix_{i=1,2,\cdots 5}} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_5 \\ v_5 \end{bmatrix} \tag{3-7}$$

Given the computed detected marker's corners and center ($m^{\tau}_{pix_{i=1,2,\cdots 5}}$) expressed in pixel coordinate values the corresponding ($2D$) image coordinates ($m^{\tau}_{2D_{i=1,2\cdots 5}}$) are computed as follows

$$m^{\tau}_{2D_{i=1,2\cdots 5}} = \begin{bmatrix} x_{2D_1} \\ y_{2D_1} \\ x_{2D_2} \\ y_{2D_2} \\ \vdots \\ x_{2D_5} \\ y_{2D_5} \end{bmatrix} = \begin{bmatrix} \frac{v_1 - c_x}{f_x} \\ \frac{u_1 - c_y}{f_y} \\ \frac{v_2 - c_x}{f_x} \\ \frac{u_2 - c_y}{f_y} \\ \vdots \\ \frac{v_5 - c_x}{f_x} \\ \frac{u_5 - c_y}{f_y} \end{bmatrix} \tag{3-8}$$

To conclude given an acquired image a detector is used to retrieve the center of the marker (u_5, v_5) and the side of it in pixel coordinate ($side_{pix}$). Given this two information available only if a detection occurs ($measurement = true$) both the detected pixel coordinates ($m^{\tau}_{pix_{i=1,2\cdots 5}}$) and ($2D$) image coordinate ($m^{\tau}_{2D_{i=1,2\cdots 5}}$) values are calculated. A scheme summarizing how to calculate $(m^{\tau}_{pix_{i=1,2\cdots 5}}, m^{\tau}_{2D_{i=1,2\cdots 5}})$ is provided for the sake of clarity.

$$\begin{bmatrix} {}^{f}image, {}^{f}ID^{\star} \\ {}^{f}f_x, {}^{f}f_y, {}^{f}c_x, {}^{f}c_y \end{bmatrix} \searrow$$
$$chosen\_camera \\ = 0 \; or \; 1 \rightarrow \begin{bmatrix} image \\ ID^{\star} \\ f_x \\ f_y \\ c_x \\ c_y \end{bmatrix} \rightarrow \begin{bmatrix} Detector \\ if(ID = ID^{\star}) \end{bmatrix} \rightarrow \begin{bmatrix} u_5 \\ v_5 \\ side_{pix} \\ f_x \\ f_y \\ c_x \\ c_y \end{bmatrix} \rightarrow \begin{bmatrix} m^{\tau}_{pix_{i=1,2\cdots 5}} \\ m^{\tau}_{2D_{i=1,2\cdots 5}} \end{bmatrix}$$
$$\begin{bmatrix} {}^{b}image, {}^{b}ID^{\star} \\ {}^{b}f_x, {}^{b}f_y, {}^{b}c_x, {}^{b}c_y \end{bmatrix} \nearrow$$

3. $(\mathbf{d^{\star}_m}, \mathbf{d_m})$: They are the desired and the estimated distance between marker and chosen camera. To estimate the distance it has been assumed that the marker is planar which means that all the marker detected points lie on the same plane and therefore they all have the same distance from the chosen camera. Using perspective projection knowing the absolute focal length (f_{mm}), the sensor height ($s_{h_{mm}}$), the number of rows of the image ($im_{h_{pix}}$), the real side of the marker in millimeters ($side_{mm}$) and the detected side of the marker in pixel ($side_{pix}$) is possible to obtain a rough distance estimation. The latter is given by

$$d_{mm} = \frac{f_{mm} \;\; side_{mm} \;\; im_{h_{pix}}}{side_{pix} \;\; s_{h_{mm}}} \quad d_m = \frac{d_{mm}}{1000} \tag{3-9}$$

A detailed explanation of how Eq. (3-9) is derived it is given in Appendix C-1. To solve the image based visual servo problem either the desired distance or an estimation of it in meters is required.

4. **measurement** : It is a boolean variable that is set to true if the desired marker has been detected successfully and false otherwise. To be precise given an image acquired by the chosen quadrotor's camera only if the desired ArUco marker with (ID) equal to $(ID^{\star})$ is detected the variable $(measurement)$ is set to true. The latter becomes false if in the sequent acquired image no detection occurs.

5. **safety_counter** : It is an integer variable that counts how many consequently not detection occurs. Every time a detection occurs the variable is set to zero. The $(safety\_counter)$ variable has been introduced to avoid that if the desired ArUco marker is lost the quadrotor will continue to move in a direction that might bring it far away from the area in which the desired ArUco marker is placed. To be precise a threshold variable has been introduced called $(safety\_counter\_threshold)$ having the goal to put the quadrotor to hover (reference velocities equal to zero) if $(safety\_counter)$ is greater than $(safety\_counter\_threshold)$. The choice to introduce this variable is mainly related to the image state estimator module. Indeed, having an estimation of the marker's corners is important to avoid problems when the detection is lost just for some instant. However, a wrong estimation might cause the quadrotor to move in a direction in which it will never see again the desired marker. Thus, the $(safety\_counter\_threshold)$ variable is used by the image based visual servo module to understand that the image state estimator module is estimating from a lot of time the corners and this might be a problem because the desired marker can be far away from the correct position. Therefore, the $(safety\_counter)$ variable can be thought as a variable able to tell to the image based visual servo module how long the quadrotor can continue to move when a detection is not available before to stop the quadrotor and wait until a detection will occur again.

6. $\mathbf{R}_{\mathbf{C}}^{\mathbf{R}}$: It represents the rotation matrix describing how the camera frame is rotated with respect to robot frame. This rotation matrix is fixed because the quadrotor's front and bottom camera do not move with respect to the robot frame fixed with the quadrotor itself. The choice to set this matrix in the perception module is motivated by the fact that every time a quadrotor camera is defined it is required to evaluate not only the default camera parameters but also the transformation between camera frame and robot frame. The latter will be used to map the image based visual servo controller output representing the velocities $(\mathbf{v}_C^{C,O})$ of the camera frame (C) with respect to the visual marker frame (O) expressed in camera coordinate frame (C) into the velocities $(\mathbf{v}_C^{R,O})$ of the camera frame (C) with respect to the visual marker frame (O) expressed in robot coordinate frame (R).

7. $(\mathbf{f_x}, \mathbf{f_y}, \mathbf{c_x}, \mathbf{c_y})$: The are the intrinsic camera coefficients of the chosen camera required to move from pixel coordinates to $(2D)$ image coordinates and vice versa. This parameters will be used in the image state estimator module to transform the estimated pixel coordinates $(\hat{m}^{\tau^{\star}}_{pix_{i=1,2,\cdots 5}})$ into the corresponding $(2D)$ estimated image coordinates $(\hat{m}^{\tau^{\star}}_{2D_{i=1,2,\cdots 5}})$.

3-1-3 Algorithm

A summary of the overall algorithm running inside the perception module is presented.

Algorithm 7 Perception Module

Initialization:

Get default variables front camera:
$({}^{f}u_5^{\star}, {}^{f}v_5^{\star}, {}^{f}d_{mm}^{\star}, {}^{f}side_{mm}, {}^{f}ID^{\star}, {}^{f}f_{mm}, {}^{f}s_{h_{mm}}, {}^{f}im_{h_{pix}}, R_{C_f}^{R})$
Get default variables bottom camera:
$({}^{b}u_5^{\star}, {}^{b}v_5^{\star}, {}^{b}d_{mm}^{\star}, {}^{b}side_{mm}, {}^{b}ID^{\star}, {}^{b}f_{mm}, {}^{b}s_{h_{mm}}, {}^{b}im_{h_{pix}}, R_{C_b}^{R})$

Algorithm: (Run at acquired image frame rate (30 Hz))

Get camera bottom image and Get front camera image
Get bottom camera intrinsic matrix $({}^{b}f_x, {}^{b}f_y, {}^{b}c_x, {}^{b}c_y)$ and distortion coefficients $({}^{b}k_1, {}^{b}k_2, {}^{b}p_1, {}^{b}p_2, {}^{b}k_3)$
Get front camera intrinsic matrix $({}^{f}f_x, {}^{f}f_y, {}^{f}c_x, {}^{f}c_y)$ and distortion coefficients $({}^{f}k_1, {}^{f}k_2, {}^{f}p_1, {}^{f}p_2, {}^{f}k_3)$
Get chosen_camera (0=bottom, 1=front)
if chosen_camera = 0 **then**
  Set img = bottom image
  Set camera intrinsic matrix $(f_x, f_y, c_x, c_y) = ({}^{b}f_x, {}^{b}f_y, {}^{b}c_x, {}^{b}c_y)$ and distortion coefficients $(k_1, k_2, p_1, p_2, k_3) = ({}^{b}k_1, {}^{b}k_2, {}^{b}p_1, {}^{b}p_2, {}^{b}k_3)$
  Set default variables $(u_5^{\star}, v_5^{\star}, d_{mm}^{\star}, side_{mm}, ID^{\star}, f_{mm}, s_{h_{mm}}, im_{h_{pix}}, R_C^R) =$
  $= ({}^{b}u_5^{\star}, {}^{b}v_5^{\star}, {}^{b}d_{mm}^{\star}, {}^{b}side_{mm}, {}^{b}ID^{\star}, {}^{b}f_{mm}, {}^{b}s_{h_{mm}}, {}^{b}im_{h_{pix}}, R_{C_b}^{R})$
else
  Set img = front image
  Set camera intrinsic matrix $(f_x, f_y, c_x, c_y) = ({}^{f}f_x, {}^{f}f_y, {}^{f}c_x, {}^{f}c_y)$ and distortion coefficients $(k_1, k_2, p_1, p_2, k_3) = ({}^{f}k_1, {}^{f}k_2, {}^{f}p_1, {}^{f}p_2, {}^{f}k_3)$
  Set default variables $(u_5^{\star}, v_5^{\star}, d_{mm}^{\star}, side_{mm}, ID^{\star}, f_{mm}, s_{h_{mm}}, im_{h_{pix}}, R_C^R) =$
  $= ({}^{f}u_5^{\star}, {}^{f}v_5^{\star}, {}^{f}d_{mm}^{\star}, {}^{f}side_{mm}, {}^{f}ID^{\star}, {}^{f}f_{mm}, {}^{f}s_{h_{mm}}, {}^{f}im_{h_{pix}}, R_{C_f}^{R})$
end if
Detect ArUco marker from img
if ArUco marker detected **then**
  Extract $(u_5, v_5, side_{pix}, ID)$
  if $(ID^* = ID)$ **then**
    measurement = true and safety_counter = 0
    Compute measurements: $(m_{pix_{i=1,2\cdots5}}^{\tau}, m_{2D_{i=1,2\cdots5}}^{\tau}) = \mathbf{f}(u_5, v_5, side_{pix}, f_x, f_y, c_x, c_y)$
    Estimate distance: $d_m = \mathbf{f}(side_{pix}, side_{mm}, f_{mm}, s_{h_{mm}}, im_{h_{pix}})$
    Send $(m_{pix_{i=1,2\cdots5}}^{\tau})$, $(m_{2D_{i=1,2\cdots5}}^{\tau})$ and (d_m)
  else
    measurement = false and safety_counter = safety_counter +1
  end if
end if
Compute references:
$(m_{pix_{i=1,2,\cdots5}}^{\tau^{\star}}, m_{2D_{i=1,2\cdots5}}^{\tau^{\star}}) = \mathbf{f}(u_5^{\star}, v_5^{\star}, d_{mm}^{\star}, side_{mm}, f_{mm}, s_{h_{mm}}, im_{h_{pix}}, f_x, f_y, c_x, c_y)$
Send $(m_{pix_{i=1,2\cdots5}}^{\tau^{\star}}), (m_{2D_{i=1,2\cdots5}}^{\tau^{\star}}), (d_m^{\star}), (\text{measurement}), (\text{safety\_counter}), (R_C^R), (f_x, f_y, c_x, c_y)$

In the upper algorithm the world *Get* is used to indicate that the perception module will read a certain input whereas the word *Send* is used to say that the perception module will output a certain value. A resume of the inputs and outputs of the perception module is provided.

$$\left[\begin{array}{c}\textbf{INPUTS}\\ 1)Default\ variables\ front\ camera\ :\\ \underline{({}^{\mathbf{f}}\mathbf{u}^{\star}_{\mathbf{5}}, {}^{\mathbf{f}}\mathbf{v}^{\star}_{\mathbf{5}}, {}^{\mathbf{f}}\mathbf{d}^{\star}_{\mathbf{mm}}, {}^{\mathbf{f}}\mathbf{side}_{\mathbf{mm}}, {}^{\mathbf{f}}\mathbf{ID}^{\star},}\\ \underline{{}^{\mathbf{f}}\mathbf{f}_{\mathbf{mm}}, {}^{\mathbf{f}}\mathbf{s}_{\mathbf{h_{mm}}}, {}^{\mathbf{f}}\mathbf{im}_{\mathbf{h_{pix}}}, \mathbf{R}^{\mathbf{R}}_{\mathbf{C_f}})}\\ 2)Default\ variables\ bottom\ camera\ :\\ \underline{({}^{\mathbf{b}}\mathbf{u}^{\star}_{\mathbf{5}}, {}^{\mathbf{b}}\mathbf{v}^{\star}_{\mathbf{5}}, {}^{\mathbf{b}}\mathbf{d}^{\star}_{\mathbf{mm}}, {}^{\mathbf{b}}\mathbf{side}_{\mathbf{mm}}, {}^{\mathbf{b}}\mathbf{ID}^{\star},}\\ \underline{{}^{\mathbf{b}}\mathbf{f}_{\mathbf{mm}}, {}^{\mathbf{b}}\mathbf{s}_{\mathbf{h_{mm}}}, {}^{\mathbf{b}}\mathbf{im}_{\mathbf{h_{pix}}}, \mathbf{R}^{\mathbf{R}}_{\mathbf{C_b}})}\\ 3)\underline{\mathbf{chosen\_camera}}\ (0 = bottom, 1 = front)\\ 4)\underline{\mathbf{Front\ camera\ image}}\\ 5)\underline{\mathbf{Bottom\ camera\ image}}\\ 6)Bottom\ camera\ intrinsic\ matrix\\ and\ distortion\ coefficients\ :\\ \underline{({}^{\mathbf{b}}\mathbf{f}_{\mathbf{x}}, {}^{\mathbf{b}}\mathbf{f}_{\mathbf{y}}, {}^{\mathbf{b}}\mathbf{c}_{\mathbf{x}}, {}^{\mathbf{b}}\mathbf{c}_{\mathbf{y}}, {}^{\mathbf{b}}\mathbf{k}_{\mathbf{1}}, {}^{\mathbf{b}}\mathbf{k}_{\mathbf{2}}, {}^{\mathbf{b}}\mathbf{p}_{\mathbf{1}}, {}^{\mathbf{b}}\mathbf{p}_{\mathbf{2}}, {}^{\mathbf{b}}\mathbf{k}_{\mathbf{3}})}\\ 7)Front\ camera\ intrinsic\ matrix\\ and\ distortion\ coefficients\ :\\ \underline{({}^{\mathbf{f}}\mathbf{f}_{\mathbf{x}}, {}^{\mathbf{f}}\mathbf{f}_{\mathbf{y}}, {}^{\mathbf{f}}\mathbf{c}_{\mathbf{x}}, {}^{\mathbf{f}}\mathbf{c}_{\mathbf{y}}, {}^{\mathbf{f}}\mathbf{k}_{\mathbf{1}}, {}^{\mathbf{f}}\mathbf{k}_{\mathbf{2}}, {}^{\mathbf{f}}\mathbf{p}_{\mathbf{1}}, {}^{\mathbf{f}}\mathbf{p}_{\mathbf{2}}, {}^{\mathbf{f}}\mathbf{k}_{\mathbf{3}})}\end{array}\right] \rightarrow \left[\begin{array}{c}\textbf{OUTPUTS}\\ 1)Desired\ pixel\ coordinates\ :\\ \underline{(\mathbf{m}^{\tau^{\star}}_{\mathbf{pix_{i=1,2,\cdots 5}}})}\\ 2)Detected\ pixel\ coordinates\ :\\ \underline{(\mathbf{m}^{\tau}_{\mathbf{pix_{i=1,2\cdots 5}}})}\\ 3)Desired\ 2D\ image\ coordinates\ :\\ \underline{(\mathbf{m}^{\tau^{\star}}_{\mathbf{2D_{i=1,2\cdots 5}}})}\\ 4)Detected\ 2D\ image\ coordinates\ :\\ \underline{(\mathbf{m}^{\tau}_{\mathbf{2D_{i=1,2\cdots 5}}})}\\ 5)Desired\ distance\ in\ meters\ between\\ chosen\ camera\ frame\ and\ marker\\ (Assumption\ Marker\ is\ planar)\ :\ \underline{\mathbf{d}^{\star}_{\mathbf{m}}}\\ 6)Estimated\ distance\ in\ meters\ between\\ chosen\ camera\ frame\ and\ marker\\ (Assumption\ Marker\ is\ planar)\ :\ \underline{\mathbf{d}_{\mathbf{m}}}\\ 7)True/False\ detection\ variable\ :\\ \underline{\mathbf{measurement}}\\ 8)Safety\ variable\ counting\ the\ number\\ of\ consequently\ no\ detection\ :\\ \underline{\mathbf{safety\_counter}}\\ 9)Fixed\ rotation\ matrix\ from\\ camera\ frame\ to\ robot\ frame\ :\ \underline{\mathbf{R}^{\mathbf{R}}_{\mathbf{C}}}\\ 10)Intrinsic\ matrix\ :\ \underline{(\mathbf{f}_{\mathbf{x}}, \mathbf{f}_{\mathbf{y}}, \mathbf{c}_{\mathbf{x}}, \mathbf{c}_{\mathbf{y}})}\end{array}\right]$$

3-2 Image state estimator module

The image state estimator module is in charge of estimating the pixel points (marker's corners and center) at a certain desired chosen frequency which will be the same of the image based visual servo controller module. The need of this module is mainly due to the fact that the image based visual servo controller module has to generate the output in a synchronous way which means at a specific frequency rate. Thus it cannot work using as a feedback the marker's corners and center detected by the perception module because it is not guaranteed that every time an image is acquired a detection is provided. Indeed something can appear between marker and camera and the detected pixel points might not be available at one specific discrete time instant which means that a feedback is not available for the image based visual servo controller module. For this reason to ensure that the corners are provided at a chosen frequency rate the image state estimator module has been introduced. Two different approach to provide an estimation of the detected pixel points when a detection is not available have been developed.

- **Static Approach:** This approach when a detection is not available it assigns to the estimated pixel points the values of the previous detected ones. This approach does

not try to guess how the pixel points moved on the image plane if the marker for some reason has disappeared from the image.

- **Kalman Filter Approach:** This approach uses a Kalman filter with a velocity constant model to estimate how the pixel points (marker's corners and center) have been moved on the image plane when the detected pixel points are not available. The aim of this filter is to estimate how the pixel points move on the image plane to try to guess how the marker is moving in space.

According to different tasks one approach might result better than the other. Indeed, in a scenario where the marker does not move it is better to use the static approach because if the marker disappear it is because something has appeared between camera and marker. Therefore, it is better to avoid to guess where the marker is because it is already known that the marker is in the same position of before. Thus, in this situation it is better to assign to the estimated pixel points the previous detected values to avoid that the quadrotor will move far way from the desired area where the marker is located. On the other hand in dealing with a marker that is moving freely in space it is better to try to estimate the pixel points to guess where the marker is when it disappears from the image. In this case there is not advantage in keeping the estimated pixel points values equal to the previous detected ones because the marker is not static anymore and it is position in space might randomly change.

3-2-1 Inputs

1. $(\mathbf{measurement}), (\mathbf{m}^{\tau}_{\mathbf{pix_{i=1,2\cdots 5}}}, \mathbf{m}^{\tau}_{\mathbf{2D_{i=1,2\cdots 5}}}), (\mathbf{f_x}, \mathbf{f_y}, \mathbf{c_x}, \mathbf{c_y})$: They are the outputs of the perception module. The meaning of them have been already discussed in subsection 3-1-2.

2. **selector** : This is an integer variable that is used to switch among the static approach (selector=0) and the Kalman filter approach (selector=1).
   By default the selector is set to one. Different task can required different approaches therefore this parameter can be changed online.

3. $\mathbf{Q}, \mathbf{R}, \mathbf{T_s}$: They represent the process covariance noise matrix (Q), the measurement covariance noise matrix (R) and the sampling time (T_s) at which the image is acquired. A reasonable sampling time choice is ($T_s = 1/30 = 0.033s$) that is derived assuming that the camera provides images at a frequency of $30Hz$. It is also possible to increase and decrease the variable (T_s) to a have bigger or a smaller displacement of the predicted pixel points between two consequently acquired images. Indeed increasing (T_s) the estimated corners in the second image will appear further than in the case in which (T_s) is decreased. Choosing ($T_s = 0$) will make the estimated corners to do not move on the image plane which resembles to the static filter approach. The process covariance (Q) and measurement noise (R) matrices are used to tell to the Kalman filter respectively how much the noise affects the states and how much the noise affects the measurements.

The (Q) and (R) matrices are defined as

$$
\begin{aligned}
Q_{20\times 20} &= \begin{bmatrix} Q_{T4\times 4} & 0_{4\times 4} & 0_{4\times 4} & 0_{4\times 4} & 0_{4\times 4} \\ 0_{4\times 4} & Q_{T4\times 4} & 0_{4\times 4} & 0_{4\times 4} & 0_{4\times 4} \\ 0_{4\times 4} & 0_{4\times 4} & Q_{T4\times 4} & 0_{4\times 4} & 0_{4\times 4} \\ 0_{4\times 4} & 0_{4\times 4} & 0_{4\times 4} & Q_{T4\times 4} & 0_{4\times 4} \\ 0_{4\times 4} & 0_{4\times 4} & 0_{4\times 4} & 0_{4\times 4} & Q_{T4\times 4} \end{bmatrix} \\
R_{10\times 10} &= \begin{bmatrix} R_{T2\times 2} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & R_{T2\times 2} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & R_{T2\times 2} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} & R_{T2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} & R_{T2\times 2} \end{bmatrix}
\end{aligned}
\tag{3-10}
$$

where $(Q_{T4\times 4})$ and $(R_{T4\times 4})$ are the two diagonal covariance matrices used to tune the Kalman filter. The values of these matrices are usually provided as default parameters of the Kalman filter and they are usually loaded in the initialization of the Kalman filter itself. The size of (Q) and (R) matrices is determined by the states and measurement size. Indeed at a specific discrete time instant (k) the estimated states $(\hat{x}(k))$ and the measurement $(z(k))$ (if they are available) are defined as

$$
\begin{aligned}
\hat{x}(k) &= \begin{bmatrix} \hat{u}_1(k) & \hat{v}_1(k) & \hat{\dot{u}}_1(k) & \hat{\dot{v}}_1(k) & \cdots & \hat{u}_5(k) & \hat{v}_5(k) & \hat{\dot{u}}_5(k) & \hat{\dot{v}}_5(k) \end{bmatrix}^T_{20\times 20} \\
z(k) &= m^{\tau}_{pix_{i=1,2\cdots 5}}(k) = \begin{bmatrix} u_1(k) & v_1(k) & u_2(k) & v_2(k) & \cdots & u_5(k) & v_5(k) \end{bmatrix}^T_{10\times 10}
\end{aligned}
\tag{3-11}
$$

In tuning the (Q) and (R) covariance matrices the following consideration have been done.

- **Q**: The covariance process noise matrix (Q) matrix takes into account uncertainty in the model. Increasing the diagonal elements of (Q) causes the Kalman filter prediction to be far from the provided model's value. This occurs because the process covariance noise (Q) is added to the process model matrix (A) and large values in (Q) results in large noise added inside the model on which the Kalman filter rely on. Therefore if it is known that the pixel points will move following a velocity constant model is better to keep (Q) small. If instead the visual marker is moving randomly it is better to increase (Q) to tell to the Kalman filter that it does not have to rely too much on the model available. In choosing a diagonal matrix it is assumed that the states are independent from each other.
- **R**: The measurement covariance matrix (R) matrix takes into account uncertainty in the measurements. Increasing the diagonal elements of (R) more uncertainty is introduced in the measurements. This causes the predicted pixel points to converge slower towards the real measurements ones because the Kalman filter does not rely too much on the measurements values. On the other hand decreasing the diagonal element of the (R) matrix means that the Kalman filter in making the prediction will rely more on the measurements rather than on the model. This choice will cause the predicted values to converge faster towards the detected measurements ones when they are available.

3-2-2 Outputs

1. $\underline{(\hat{\mathbf{m}}^{\tau}_{\mathbf{pix_{i=1,2\cdots5}}}, \hat{\mathbf{m}}^{\tau}_{\mathbf{2D_{i=1,2\cdots5}}})}$: They represents the estimated corners in both pixels and $(2D)$ image coordinates. The need of a corners' estimation is mainly due to the fact that in the attempt of following an object where the visual marker is placed on it the quadrotor can loose the marker due to unknowns and fast movements of the object. For this reason a corners' estimator has been implemented aiming at predict the corners' movement when a detection is not available. Furthermore, a detection is provided at a certain detector frequency which might be a feedback to slow for the image based visual servo controller. Therefore, it is a good design choice to introduce an estimator able to select the frequency at which the corners are needed. In using for example a Kalman filter to estimate the corners is important to be aware that the filter will predict the corners pixel values if no detection occurs and it will update the prediction with real detected corners pixel values every time a detection occurs. This means that it is possible to generate predicted corners at every desired frequency but to be able to rely on them to solve a landing or object following task it is better to select the state estimator module frequency around $30Hz$ to ensure that the number of corners prediction is not to high with respect to then number of detections. To make the image based visual servo controller synchronous which means that it will work at a desired chosen frequency and not only when a detection is available two possible solutions have been developed.

   - **Static approach**: It consists in assigning to the estimated pixel points and $(2D)$ image coordinates values $(\hat{m}^{\tau}_{pix_{i=1,2\cdots5}}, \hat{m}^{\tau}_{2D_{i=1,2\cdots5}})$ the detected ones $(m^{\tau}_{pix_{i=1,2\cdots5}}$, $m^{\tau}_{2D_{i=1,2\cdots5}})$ when they are available otherwise it keeps the previous detected values until a new one is provided by the perception module. This approach allows the image based visual servo controller module to work having feedbacks at a chosen frequency. However this approach does not provide any information on how the corners move on the image plane if a detection is not available anymore. Applying this approach, unexpected movements of the visual marker might result in a lost of the marker from the image plane which will make the quadrotor unable to complete the vision based planner task. A resume of the algorithm is provided.

Algorithm 8 Static approach

Initialization:

Initialize previous values $({}^{prev}m^{\tau}_{pix_{i=1,2,\cdots5}}, {}^{prev}m^{\tau}_{2D_{i=1,2,\cdots5}}) = (0_{10\times1}, 0_{10\times1})$

Algorithm: (Run at desired frequency)

Get measurement
if measurement = true **then**
  Get $(m^{\tau}_{pix_{i=1,2,\cdots5}}, m^{\tau}_{2D_{i=1,2,\cdots5}})$
  $({}^{prev}m^{\tau}_{pix_{i=1,2,\cdots5}}, {}^{prev}m^{\tau}_{2D_{i=1,2,\cdots5}}) = (m^{\tau}_{pix_{i=1,2,\cdots5}}, m^{\tau}_{2D_{i=1,2,\cdots5}})$
  $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots5}}, \hat{m}^{\tau}_{2D_{i=1,2,\cdots5}}) = (m^{\tau}_{pix_{i=1,2,\cdots5}}, m^{\tau}_{2D_{i=1,2,\cdots5}})$
else
  $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots5}}, \hat{m}^{\tau}_{2D_{i=1,2,\cdots5}}) = ({}^{prev}m^{\tau}_{pix_{i=1,2,\cdots5}}, {}^{prev}m^{\tau}_{2D_{i=1,2,\cdots5}})$
end if
Send $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots5}}, \hat{m}^{\tau}_{2D_{i=1,2,\cdots5}})$

- **Kalman filter approach:** It consists in using a Kalman filter to estimate the corners and the center of the chosen ArUco marker. This approach will provide an estimation of how the marker's corners and the center move on the image plane if a detection is a not available anymore. The Kalman filter formulation is given in [9]. To design the Kalman filter the process and the measurement matrices are set based on a velocity constant model. The input of the Kalman filter are the detected pixel points ($m^{\tau}_{pix_{i=1,2\cdots5}}(k)$) at discrete time instant (k).
  The process velocity constant model given a generic pixel point (i) is given by

$$\begin{aligned}\hat{u}_i(k) &= \hat{u}_i(k-1) + T_s\hat{\dot{u}}_i(k-1)\\ \hat{v}_i(k) &= \hat{v}_i(k-1) + T_s\hat{\dot{v}}_i(k-1)\\ \hat{\dot{u}}_i(k) &= \hat{\dot{u}}_i(k-1)\\ \hat{\dot{v}}_i(k) &= \hat{\dot{u}}_i(k-1)\end{aligned} \tag{3-12}$$

  Extending the model of a single pixel point (i) to the marker's corners and center it is possible to derive the process model used in the image state estimator module to estimate the position in pixel and the velocity of the marker's corners and center. The process model is

$$\hat{x}(k) = A\hat{x}(k-1) \tag{3-13}$$

  which is equivalent to

$$\begin{bmatrix}\hat{u}_1(k)\\ \hat{v}_1(k)\\ \hat{\dot{u}}_1(k)\\ \hat{\dot{v}}_1(k)\\ \hat{u}_2(k)\\ \hat{v}_2(k)\\ \hat{\dot{u}}_2(k)\\ \hat{\dot{v}}_2(k)\\ \vdots\\ \hat{u}_5(k)\\ \hat{v}_5(k)\\ \hat{\dot{u}}_5(k)\\ \hat{\dot{v}}_5(k)\end{bmatrix} = \begin{bmatrix}A_{T4\times4} & 0 & 0 & 0 & 0\\ 0 & A_{T4\times4} & 0 & 0 & 0\\ 0 & 0 & A_{T4\times4} & 0 & 0\\ 0 & 0 & 0 & A_{T4\times4} & 0\\ 0 & 0 & 0 & 0 & A_{T4\times4}\end{bmatrix} \begin{bmatrix}\hat{u}_1(k-1)\\ \hat{v}_1(k-1)\\ \hat{\dot{u}}_1(k-1)\\ \hat{\dot{v}}_1(k-1)\\ \hat{u}_2(k-1)\\ \hat{v}_2(k-1)\\ \hat{\dot{u}}_2(k-1)\\ \hat{\dot{v}}_2(k-1)\\ \vdots\\ \hat{u}_5(k-1)\\ \hat{v}_5(k-1)\\ \hat{\dot{u}}_5(k-1)\\ \hat{\dot{v}}_5(k-1)\end{bmatrix} \tag{3-14}$$

  with

$$A_{T4\times4} = \begin{bmatrix}1 & 0 & T_s & 0\\ 0 & 1 & 0 & T_s\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3-15}$$

  The measurement model is

$$m^{\tau}_{pix_{i=1,2\cdots5}}(k) = Cm^{\tau}_{pix_{i=1,2\cdots5}}(k-1) \rightarrow z(k) = Cz(k-1) \tag{3-16}$$

that is equivalent to:

$$\begin{bmatrix} u_1(k) \\ v_1(k) \\ u_2(k) \\ v_2(k) \\ \vdots \\ u_5(k) \\ v_5(k) \end{bmatrix} = \begin{bmatrix} C_{T2\times4} & 0 & 0 & 0 & 0 \\ 0 & C_{T2\times4} & 0 & 0 & 0 \\ 0 & 0 & C_{T2\times4} & 0 & 0 \\ 0 & 0 & 0 & C_{T2\times4} & 0 \\ 0 & 0 & 0 & 0 & C_{T2\times4} \end{bmatrix} \begin{bmatrix} u_1(k-1) \\ v_1(k-1) \\ u_2(k-1) \\ v_2(k-1) \\ \vdots \\ u_5(k-1) \\ v_5(k-1) \end{bmatrix} \tag{3-17}$$

with

$$C_{T2\times4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3-18}$$

The Kalman filter approach with its predict and update stage is summarized here.

Algorithm 9 Kalman filter approach

Initialization:

Get default variables: $(Q, R, T_s = 0.033)$
Calculate $A = \mathbf{f}(T_s)$, C matrices

Algorithm: (Run at desired frequency)

Get intrinsic matrix (f_x, f_y, c_x, c_y)
Get measurement
if measurement = true (update Kalman filter stage) **then**
  Get $(m^{\tau}_{pix_{i=1,2,\cdots5}})$
  if first time (first time the marker is detected) **then**
    Initialize Kalman filter: $\hat{x} = \begin{bmatrix} u_1 & v_1 & T_s & T_s \cdots u_5 & v_5 & T_s & T_s \end{bmatrix}^T_{20\times1}$ and $\hat{P} = I_{20\times20}$ (Identity matrix)
  end if
  Set $z = m^{\tau}_{pix_{i=1,2,\cdots5}}$
  $\hat{P}_{model} = A\hat{P}A^T + Q$
  $\hat{x}_{model} = A\hat{x}$
  $K = (\hat{P}_{model}C^T)(R + C\hat{P}_{model}C^T)^{-1}$
  $\hat{P} = \hat{P}_{model} - KC\hat{P}_{model}$
  $\hat{x} = \hat{x}_{model} + K(z - C\hat{x}_{model})$ (Update)
else
  $\hat{P} = \hat{P}_{model} = A\hat{P}A^T + Q$ (Prediction covariance)
  $\hat{x} = \hat{x}_{model} = A\hat{x}$ (Prediction states)
end if
Calculate $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots5}}) = \mathbf{f}(\hat{x}) = \begin{bmatrix} \hat{u}_1 & \hat{v}_1 & \hat{u}_2 & \hat{v}_2 & \cdots & \hat{u}_5 & \hat{v}_5 \end{bmatrix}^T$ with $\hat{x} = \begin{bmatrix} \hat{u}_1 & \hat{v}_1 & \hat{\dot{u}}_1 & \hat{\dot{v}}_1 & \cdots & \hat{u}_5 & \hat{v}_5 & \hat{\dot{u}}_5 & \hat{\dot{v}}_5 \end{bmatrix}^T$
Calculate $(\hat{m}^{\tau}_{2D_{i=1,2,\cdots5}}) = \mathbf{f}(\hat{m}^{\tau}_{pix_{i=1,2,\cdots5}}, f_x, f_y, c_x, c_y)$
Send $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots5}}, \hat{m}^{\tau}_{2D_{i=1,2,\cdots5}})$

3-2-3 Algorithm

A summary of the overall algorithm running inside the image state estimator module is provided.

Algorithm 10 Image state estimator module

Initialization:

Get selector (0= Static approach, 1 Kalman filter approach)
Get default variables: $(Q, R, T_s = 0.033)$
Calculate $A = \mathbf{f}(T_s)$, C matrices
Initialize previous values $({}^{prev}m^{\tau}_{pix_{i=1,2,\cdots 5}}, {}^{prev}m^{\tau}_{2D_{i=1,2,\cdots 5}}) = (0_{10\times 1}, 0_{10\times 1})$

Algorithm: (Run at image frequency 30Hz to have as much measurements as possible)

Get measurement
Get intrinsic matrix (f_x, f_y, c_x, c_y)
if selector = 0 **then**
  if measurement = true **then**
    Get $(m^{\tau}_{pix_{i=1,2,\cdots 5}}, m^{\tau}_{2D_{i=1,2,\cdots 5}})$
    $({}^{prev}m^{\tau}_{pix_{i=1,2,\cdots 5}}, {}^{prev}m^{\tau}_{2D_{i=1,2,\cdots 5}}) = (m^{\tau}_{pix_{i=1,2,\cdots 5}}, m^{\tau}_{2D_{i=1,2,\cdots 5}})$
    $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots 5}}, \hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}) = (m^{\tau}_{pix_{i=1,2,\cdots 5}}, m^{\tau}_{2D_{i=1,2,\cdots 5}})$
  else
    $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots 5}}, \hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}) = ({}^{prev}m^{\tau}_{pix_{i=1,2,\cdots 5}}, {}^{prev}m^{\tau}_{2D_{i=1,2,\cdots 5}})$
  end if
else
  if measurement = true (update Kalman filter stage) **then**
    Get $(m^{\tau}_{pix_{i=1,2,\cdots 5}})$
    if first time (first time the marker is detected) **then**
      Initialize Kalman filter: $\hat{x} = \begin{bmatrix} u_1 & v_1 & T_s & T_s \cdots u_5 & v_5 & T_s & T_s \end{bmatrix}^T_{20\times 1}$ and $\hat{P} = I_{20\times 20}$ (Identity matrix)
    end if
    Set $z = m^{\tau}_{pix_{i=1,2,\cdots 5}}$ (measured variables)
    $\hat{P}_{model} = A\hat{P}A^T + Q$
    $\hat{x}_{model} = A\hat{x}$
    $K = (\hat{P}_{model}C^T)(R + C\hat{P}_{model}C^T)^{-1}$
    $\hat{P} = \hat{P}_{model} - KC\hat{P}_{model}$
    $\hat{x} = \hat{x}_{model} + K(z - C\hat{x}_{model})$
  else
    $\hat{P} = \hat{P}_{model} = A\hat{P}A^T + Q$
    $\hat{x} = \hat{x}_{model} = A\hat{x}$
  end if
  Calculate $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots 5}}) = \mathbf{f}(\hat{x}) = \begin{bmatrix} \hat{u}_1 & \hat{v}_1 & \hat{u}_2 & \hat{v}_2 & \cdots & \hat{u}_5 & \hat{v}_5 \end{bmatrix}^T$ with $\hat{x} = \begin{bmatrix} \hat{u}_1 & \hat{v}_1 & \hat{\dot{u}}_1 & \hat{\dot{v}}_1 & \cdots & \hat{u}_5 & \hat{v}_5 & \hat{\dot{u}}_5 & \hat{\dot{v}}_5 \end{bmatrix}^T$
  Calculate $(\hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}) = \mathbf{f}(\hat{m}^{\tau}_{pix_{i=1,2,\cdots 5}}, f_x, f_y, c_x, c_y)$
  Send $(\hat{m}^{\tau}_{pix_{i=1,2,\cdots 5}}, \hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}})$
end if

A resume of inputs and outputs of the image state estimator module is provided.

$$\begin{bmatrix} \textbf{INPUTS} \\ 1)\underline{\textbf{selector}} : (0 = \ Static\ approach, \\ 1 = Kalman\ filter\ approach) \\ 2)Default\ Kalman\ Filter\ Process\ and \\ Measurement\ Covariance\ Matrix : \underline{(\mathbf{Q}, \mathbf{R})} \\ 3)Sampling\ time\ acquired\ images : \underline{\mathbf{T_s}} \\ 4)True/False\ detection\ variable : \\ \underline{\textbf{measurement}} \\ 5)Detected\ pixel\ coordinates : \underline{(\mathbf{m^{\tau}_{pix_{i=1,2,\cdots 5}}})} \\ 6)Detected\ 2D\ image\ coordinates : \underline{(\mathbf{m^{\tau}_{2D_{i=1,2\cdots 5}}})} \\ 7)Intrisic : \underline{(\mathbf{f_x}, \mathbf{f_y}, \mathbf{c_x}, \mathbf{c_y})} \end{bmatrix} \rightarrow \begin{bmatrix} \textbf{OUTPUTS} \\ 1)Estimated\ pixel\ coordinates : \\ \underline{(\hat{\mathbf{m}}^{\tau}_{\mathbf{pix_{i=1,2\cdots 5}}})} \\ 2)Estimated\ 2D\ image\ coordinates : \\ \underline{(\hat{\mathbf{m}}^{\tau}_{\mathbf{2D_{i=1,2\cdots 5}}})} \end{bmatrix}$$

3-3 Image based visual servo controller module

The image based visual servo controller module is a controller having as a goal the minimization of the error between the estimated $(2D)$ image points provided by the state estimator module and the desired references ones given by the perception module. To achieve this goal it calculates the reference camera velocities with respect to the visual marker expressed in world coordinates frame that the quadrotor has to track to drive the error to zero. The formulation of the IBVS for a full actuated system is given in [10], [11], [12]. However, the quadrotor is an under-actuated system which means that it only has four control inputs (four motor speeds) to reach any $(3D)$ position in space with a certain orientation. Indeed it cannot translate along (x) and (y) axis without turning. Examples of how to apply IBVS to an under-actuated system such as the quadrotor are given in [13], [14] where the IBVS algorithm is used to solve the quadrotor autonomous landing problem. For the sake of this thesis only the instantaneous translational camera velocities $(\mathbf{v}_C^{C,O})$ will be tracked by the quadrotor. The yaw orientation is not controller by the IBVS but the yaw controller can be used to modify the quadrotor yaw angle when the IBVS algorithm is active. In modifying the yaw angle while the IBVS algorithm is active it is important to avoid to send a big change in desired yaw reference that might cause the quadrotor to rotate to fast and to loose the marker from the image. To summarize, the goal of this section is to derive an IBVS formulation that given the estimated and the desired $(2D)$ image points is able to provide the reference translational velocities required to move the quadrotor such that the error is driven to zero.

3-3-1 Inputs

1. $\underline{(\mathbf{chosen\_camera}), (\mathbf{safety\_counter}), (\mathbf{R^R_C}), (\mathbf{d_m}, \mathbf{d^{\star}_m}), (\mathbf{m^{\tau^{\star}}_{2D_{i=1,2,\cdots 5}}})}$: A deep explanation of the meaning of the perception module input variables $(chosen\_camera,\ safety\_counter,\ R^R_C,\ d^{\star}_m)$ is given in 3-1-1 whereas the calculation of the perception module output variables $(d_m, m^{\tau^{\star}}_{2D_{i=1,2,\cdots 5}})$ is given in 3-1-2.

2. $\underline{\hat{\mathbf{m}}^{\tau}_{\mathbf{2D_{i=1,2,\cdots 5}}}}$: They represent the estimated $(2D)$ visual marker's corners and center calculated by the image state estimator module either using the static approach (selector

= 0) or the Kalman filter approach (selector = 1). A detailed explanation on how these values are derived and what they represent is given in 3-2-2.

3. $\mathbf{R}_{\mathbf{R}}^{\mathbf{W}}$: It represents the rotation matrix describing how the robot frame attached with the quadrotor is rotated with respect to the world frame. A detailed explanation of how this variable is derived is given in B-1-1 and the value of the rotation matrix is given in Eq. (B-2). To calculate this matrix it is required to know the pitch (θ) , roll (ϕ) and yaw (ψ) angles of the quadrotor expressed in world coordinate frame. These values represent the orientation of the quadrotor with respect to the world coordinate frame and they are provided to the image based visual servo module either by the EKF or by a motion capture system. The latter is in charge of estimating the full quadrotor states (position, velocity, orientation ,acceleration) of the quadrotor expressed in world coordinate frame.

4. **safety_counter_threshold** : It is an integer variable that is used in combination with the ($safety\_counter$) variable whose explanation is given in 3-1-1. In particular the ($safety\_counter\_threshold$) variable has been introduced with the aim to provide to the quadrotor a threshold variable indicating after how much time it has to stop to listen to the estimation given by the image based visual servo module and wait until new commands are provided. To be precise if the ($safety\_counter$) is greater than the ($safety\_counter\_threshold$) the quadrotor will send to the navigation controller framework velocities along (x),(y),(z) direction equal to zero and not the output of the image based visual servo controller. This choice will bring the quadrotor to stay in same position (hover state). If later on the marker is detected again the velocities calculated by the image based visual servo module will be sent to the navigation controller framework.

5. ($^{\mathbf{f}}\lambda, ^{\mathbf{b}}\lambda$:) According to the chosen camera value (1=front, 0=bottom) the chosen tuning controller parameter either ($^{f}\lambda$) or ($^{b}\lambda$) is loaded into the variable (λ). The latter is a diagonal three times three positive definite matrix where the elements of its diagonal are all greater than zero .

$$\lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \quad with \quad det(\lambda) > 0 \rightarrow \lambda_1, \lambda_2, \lambda_3 > 0 \tag{3-19}$$

The values on its diagonal matrix ($\lambda_1, \lambda_2, \lambda_3$) are multiplicative always positive factors that are used to increase or decrease the velocities ($\mathbf{v}_C^{C,O}$) calculated by image based visual servo controller representing the velocities of the camera frame (C) with respect to the visual marker frame (O) expressed in camera frame coordinate (C). Therefore, the tuning parameters ($\lambda_1, \lambda_2, \lambda_3$) are applied on the velocities expressed in the chosen camera frame and not on the velocity expressed in the world frame which are the input of the navigation controller framework. This might cause misunderstanding in tuning the parameters to increase or decrease the speed of the quadrotor along a chosen quadrotor axis. Indeed, if the user wants to use the front camera to make the quadrotor approach an object and it also wants to increase the speed of the quadrotor along the (x) direction of the robot frame (assumed to be aligned with the world frame) it has to know which tuning parameter he/she has to increase to do that. For example increasing the value

of (${}^f\lambda_1$) the quadrotor will not move faster along the (x) direction of the robot frame (forward or backward) but it will move faster along the (y) direction of the quadrotor frame (left or right) (see Figure 3-4). Thus, it is important to know how the tuning parameters are related to both the velocities of the quadrotor expressed either in the robot or in the world coordinate frame. Different cameras might have different frame orientation. For this reason it is interesting how the front (${}^f\lambda_{i=1,2,3}$) and the bottom (${}^b\lambda_{i=1,2,3}$) gains are related with both the velocities of the robot expressed either in robot or world coordinate frame. Knowing this information is possible to increase or decrease the quadrotor velocities along a chosen desired direction of the world coordinate system. The navigation controller framework receives as input the velocities expressed in world coordinate frame and it has to track them. Therefore, it is required to transform the velocities derived by the IBVS controller expressed in camera coordinate frame into world coordinate frame. Knowing that the camera is fixed with the quadrotor it is possible to infer that the velocities of the camera frame is the same of the velocities of the quadrotor expressed in the chosen coordinate frame. Thus to transform the velocities of the chosen camera frame expressed in the chosen camera coordinate frame into velocities of the robot expressed in the world coordinate the following equation is used.

$$\begin{aligned} &\mathbf{v}_R^{W,O} = R_R^W \mathbf{v}_R^{R,O} \\ &with \quad \mathbf{v}_R^{R,O} = \mathbf{v}_C^{R,O} = R_C^R \mathbf{v}_C^{C,O} = R_C^R \lambda \mathbf{v}\mathbf{I}_C^{C,O} \end{aligned} \tag{3-20}$$

($\mathbf{v}\mathbf{I}_C^{C,O}$) represents the instantaneous camera velocities without any scale factor applied ($\lambda = I_{3\times3}$). Given Eq. (3-20) is possible to compute the mapping between the translational not scaled camera velocities ($\mathbf{v}\mathbf{I}_C^{C,O}$) derived using either the front (${}^{\mathbf{f}}\mathbf{v}\mathbf{I}_C^{C,O}$) or the bottom camera (${}^{\mathbf{b}}\mathbf{v}\mathbf{I}_C^{C,O}$) and the corresponding velocities expressed in robot coordinate frame (${}^{\mathbf{f}}\mathbf{v}_C^{R,O}, {}^{\mathbf{b}}\mathbf{v}_C^{R,O}$). The mapping is showed here.

$$\begin{aligned} {}^f\mathbf{v}_R^{R,O} &= {}^f\mathbf{v}_C^{R,O} = R_{C_f}^R \, {}^f\lambda \mathbf{v}\mathbf{I}_C^{C,O} = \\ &= \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} {}^f\lambda_1 & 0 & 0 \\ 0 & {}^f\lambda_2 & 0 \\ 0 & 0 & {}^f\lambda_3 \end{bmatrix} \begin{bmatrix} {}^f v_{xC}^{I^{C,O}} \\ {}^f v_{yC}^{I^{C,O}} \\ {}^f v_{zC}^{I^{C,O}} \end{bmatrix} = \begin{bmatrix} {}^f\lambda_3 {}^f v_{zC}^{I^{C,O}} \\ -{}^f\lambda_1 {}^f v_{xC}^{I^{C,O}} \\ -{}^f\lambda_2 {}^f v_{yC}^{I^{C,O}} \end{bmatrix} \end{aligned} \tag{3-21}$$

$$\begin{aligned} {}^b\mathbf{v}_R^{R,O} &= {}^b\mathbf{v}_C^{R,O} = R_{C_b}^R \, {}^b\lambda \mathbf{v}\mathbf{I}_C^{C,O} = \\ &= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} {}^b\lambda_1 & 0 & 0 \\ 0 & {}^b\lambda_2 & 0 \\ 0 & 0 & {}^b\lambda_3 \end{bmatrix} \begin{bmatrix} {}^b v_{xC}^{I^{C,O}} \\ {}^b v_{yC}^{I^{C,O}} \\ {}^b v_{zC}^{I^{C,O}} \end{bmatrix} = \begin{bmatrix} -{}^b\lambda_2 {}^f v_{yC}^{I^{C,O}} \\ -{}^b\lambda_1 {}^f v_{xC}^{I^{C,O}} \\ -{}^b\lambda_3 {}^f v_{zC}^{I^{C,O}} \end{bmatrix} \end{aligned} \tag{3-22}$$

where ($R_{C_f}^R$) and ($R_{C_b}^R$) are respectively the rotation matrix describing how the front and the bottom camera frame are rotated with respect to the robot frame. A more detailed explanation of these values in given in 3-1-1. Assuming the pitch and roll quadrotor angles are close to zero it is possible to infer that the rotation matrix (R_R^W) describing how the robot frame is rotated with respect to the world frame is only affected by the rotation of the quadrotor about the (z) quadrotor axis by a yaw (ψ) angle. Making this

assumption is possible to calculate how the tuning parameters $({}^{f}\lambda_{i=1,2,3})$ and $({}^{b}\lambda_{i=1,2,3})$ affect the velocities expressed in the world coordinate frame.

$$
\begin{aligned}
{}^{\mathbf{f}}\mathbf{v}_R^{W,O} = R_R^W {}^{f}\mathbf{v}_R^{R,O} \approx \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{f}\lambda_3 {}^{f}v_{z_C}^{I_C,O} \\ -{}^{f}\lambda_1 {}^{f}v_{x_C}^{I_C,O} \\ -{}^{f}\lambda_2 {}^{f}v_{y_C}^{I_C,O} \end{bmatrix} = \\
= \begin{bmatrix} cos(\psi) {}^{f}\lambda_3 {}^{f}v_{z_C}^{I_C,O} + sin(\psi) {}^{f}\lambda_1 {}^{f}v_{x_C}^{I_C,O} \\ sin(\psi) {}^{f}\lambda_3 {}^{f}v_{z_C}^{I_C,O} - cos(\psi) {}^{f}\lambda_1 {}^{f}v_{x_C}^{I_C,O} \\ -{}^{f}\lambda_2 {}^{f}v_{y_C}^{I_C,O} \end{bmatrix}
\end{aligned}
\tag{3-23}
$$

$$
\begin{aligned}
{}^{\mathbf{b}}\mathbf{v}_R^{W,O} = R_R^W {}^{b}\mathbf{v}_R^{R,O} \approx \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -{}^{b}\lambda_2 {}^{b}v_{y_C}^{I_C,O} \\ -{}^{b}\lambda_1 {}^{b}v_{x_C}^{I_C,O} \\ -{}^{b}\lambda_3 {}^{b}v_{z_C}^{I_C,O} \end{bmatrix} = \\
= \begin{bmatrix} -cos(\psi) {}^{b}\lambda_2 {}^{b}v_{y_C}^{I_C,O} + sin(\psi) {}^{b}\lambda_1 {}^{b}v_{x_C}^{I_C,O} \\ -sin(\psi) {}^{b}\lambda_2 {}^{b}v_{y_C}^{I_C,O} - cos(\psi) {}^{b}\lambda_1 {}^{b}v_{x_C}^{I_C,O} \\ -{}^{b}\lambda_3 {}^{b}v_{z_C}^{I_C,O} \end{bmatrix}
\end{aligned}
\tag{3-24}
$$

A table summarizing how the front $({}^{f}\lambda_{i=1,2,3})$ and the bottom $({}^{b}\lambda_{i=1,2,3})$ tuning parameters influence the velocities of the quadrotor expressed in camera ($\mathbf{v}_R^{C,O}$), robot ($\mathbf{v}_R^{R,O}$) and world ($\mathbf{v}_R^{W,O}$) coordinate frame is provided.

Table 3-1: Relation between tuning parameters $({}^{f}\lambda_1, {}^{f}\lambda_2, {}^{f}\lambda_3)$ and $({}^{b}\lambda_1, {}^{b}\lambda_2, {}^{b}\lambda_3)$ and velocities of the quadrotor expressed camera, robot and world coordinate frames

	$\mathbf{v}_C^{C,O} = \mathbf{v}_R^{C,O}$ (IBVS)	$\mathbf{v}_R^{R,O}$	$\mathbf{v}_R^{W,O}$ (Navigation controller)
${}^{f}\lambda_1$	it affects $v_{x_C}^{C,O}$	it affects $v_{y_R}^{R,O}$	It affects $v_{x_R}^{W,O}$ and $v_{y_R}^{W,O}$
${}^{f}\lambda_2$	it affects $v_{y_C}^{C,O}$	it affects $v_{z_R}^{R,O}$	it affects $v_{z_R}^{W,O}$
${}^{f}\lambda_3$	it affects $v_{z_C}^{C,O}$	it affects $v_{x_R}^{R,O}$	it affects $v_{x_R}^{W,O}$ and $v_{y_R}^{W,O}$
${}^{b}\lambda_1$	it affects $v_{x_C}^{C,O}$	it affects $v_{y_R}^{R,O}$	it affects $v_{x_R}^{W,O}$ and $v_{y_R}^{W,O}$
${}^{b}\lambda_2$	it affects $v_{y_C}^{C,O}$	it affects $v_{x_R}^{R,O}$	it affects $v_{x_R}^{W,O}$ and $v_{y_R}^{W,O}$
${}^{b}\lambda_3$	it affects $v_{z_C}^{C,O}$	it affects $v_{z_R}^{R,O}$	it affects $v_{z_R}^{W,O}$

Because the navigation controller framework receives as input the velocities of the quadrotor expressed in world coordinate frame ($\mathbf{v}_R^{W,O}$) in tuning the front camera parameters $({}^{f}\lambda_1, {}^{f}\lambda_2, {}^{f}\lambda_3)$ it is preferred to set both $({}^{f}\lambda_1, {}^{f}\lambda_3)$ around the same value if the quadrotor will perform rotation about the (z) axis. Otherwise knowing the orientation of the robot frame with respect to the world frame is possible to do a fine tuning of $({}^{f}\lambda_1, {}^{f}\lambda_3)$ to achieve a greater speed either along the (x) or (y) direction of the world coordinate frame. The same consideration referring to table Table 3-1 can be done for the bottom tuning parameters $({}^{b}\lambda_1, {}^{b}\lambda_2, {}^{b}\lambda_3)$.

6. ($^{\mathbf{f}}$**interaction_matrix**, $^{\mathbf{b}}$**interaction_matrix**) : They represent integers variables respectively associated to the front or bottom camera that are used to selected which type of interaction matrix the image based visual servo controller has to use. Given a task the image based visual servo controller has to generate the velocities that the camera frame has to track to ensure that the estimated $(2D)$ image points will coincide with the desired ones. In doing this, it can use three different types of error interaction matrices

   - *Desired error Interaction Matrix* ($interaction\_matrix = 0$): This is a constant interaction matrix that is a function of the desired center $(x^{\star}_{2D_5}, y^{\star}_{2D_5})$ of the marker,of the difference between the desired third and the desired second corner $(d^{\star}_{y_{2D_{2,3}}})$ along the (y) direction of the $(2D)$ image coordinate system and of the desired distance in meters between visual marker and chosen camera $(d^{\star}_m)$.
   - *Measured error Interaction Matrix* ($interaction\_matrix = 1$) : This interaction matrix is not constant. Indeed it changes at every iteration and it is a function of the estimated center $(\hat{x}_{2D_5}, \hat{y}_{2D_5})$, of the estimated difference between third and second corners $(\hat{d}_{y_{2D_{2,3}}})$ along the (y) direction of the $(2D)$ image coordinate system and of the estimated distance in meters between the visual marker and the chosen camera $(\hat{d}_m)$. This type of interaction matrix will generate large velocities changes which will make the quadrotor to approach the visual marker in an aggressive way.
   - *Average error Interaction Matrix* ($interaction\_matrix = 2$): It is an average interaction matrix combining both the information of the desired and of the measured interaction matrices. Choosing this interaction matrix will make the quadrotor to move faster than the desired interaction matrix but slower than the measured interaction matrix. This interaction matrix is usually preferred with respect to the other two because it is able to combine more information together to generate the velocities for the navigation controller framework.

   A detailed explanation of how these interaction matrices are derived is given in 3-2-2.

7. (**Max**, **Min**) : They are saturation constraints that are used to saturate the velocities $(\mathbf{v}_R^{W,O})$ that are sent to the navigation controller framework. A saturation should not occur tuning properly the $({}^f\lambda)$ and $({}^b\lambda)$ matrices. However for a safety condition it has been preferred to introduce this saturation constraints to avoid that the navigation controller framework will receive velocities to high to be tracked. Furthermore, the small angle approximation assumption has been used to design the controllers of the navigation controller framework. According to this assumption the quadrotor cannot track large velocities references (greater than 1 m/s) because the horizontal and vertical speed controller should not generated a pitch and roll greater than fifteen degrees to meet the small angle approximation assumption. For this reason it has been chosen to saturate the velocities at 1 m/s.

3-3-2 Outputs

1. $\mathbf{v}_R^{W,O}$: The main steps required to compute the translational camera velocities that the quadrotor has to track to minimize the error between the estimated and the desired corners are shown.

(a) **Compute Error (e)**
Given the estimated ($\hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}$) and the desired ($m^{\tau\star}_{2D_{i=1,2,\cdots 5}}$) corners expressed in ($2D$) image coordinates, following the error formulation of [15, chapter 7, pages: 189–191] is possible to write that the IBVS error (positive feedback) is given by

$$e = \mathbf{f}(\hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}, \hat{m}^{\tau\star}_{2D_{i=1,2,\cdots 5}}) = \begin{bmatrix} \hat{x}_{2D_5} - x^{\star}_{2D_5} \\ \hat{y}_{2D_5} - y^{\star}_{2D_5} \\ d_{y_{2D_{3,2}}} - d^{\star}_{y_{2D_{3,2}}} \end{bmatrix} \tag{3-25}$$

with ($d_{y_{2D_{3,2}}} = y_{2D_3} - y_{2D_2}$) and ($d^{\star}_{y_{2D_{3,2}}} = y^{\star}_{2D_3} - y^{\star}_{2D_2}$). In calculating the error the positive distance along the (y) direction between the second and the third marker has been considered. This choice is mainly due to the fact that it has been preferred to formulate the error saying that the IBVS controller has to minimize the error between the estimated and the desired center and it has to bring it is detected current side equal to the desired one. An illustration is provided showing the meaning of the error parameters.

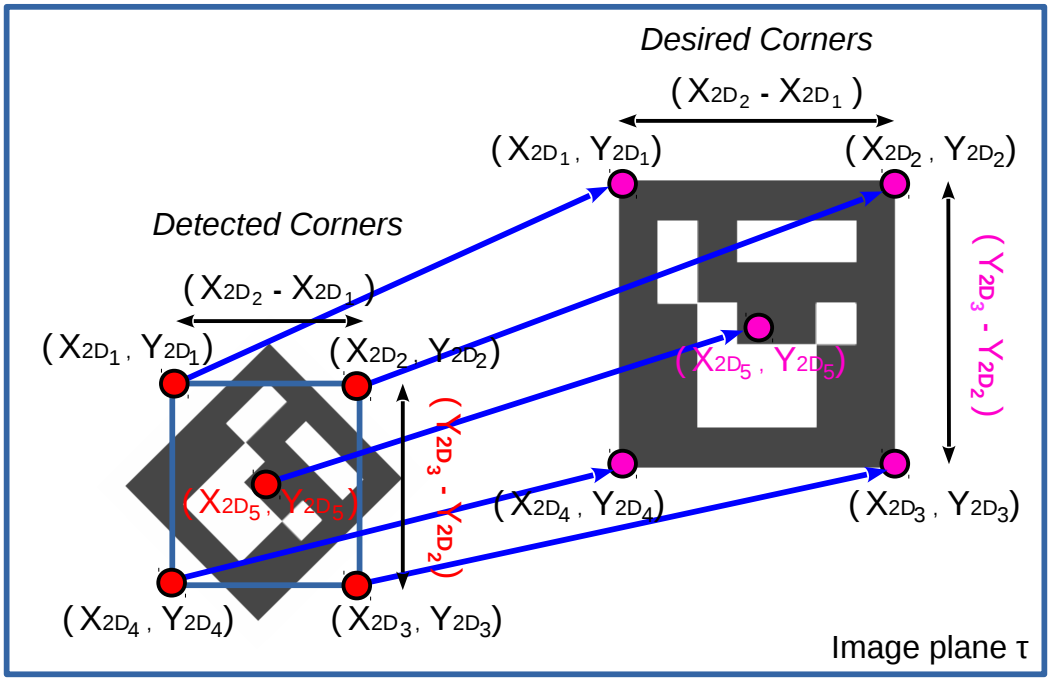


Figure 3-8: Illustration of the error terms. On the left image side the detected corners (red) are shown. They are extracted independently from the marker orientation knowing the detected center (u_5, v_5) and the side of the marker ($side_{pix}$). On the right side the desired corners (pink) are illustrated. Respectively in red and pink have been highlighted the $2D$ image coordinates values used to formulate the IBVS error.

(b) **Compute Error Interaction Matrix ($\mathbf{L_e}$)**
Dealing with visual servoing algorithm such IBVS brings immediately to underline

the concept of interaction matrix. The latter is a matrix relating the chosen visual features with the camera instantaneous velocity (translational and rotational camera frame velocities). The derivation of the interaction matrix for different visual features (point, straight line, plane primitives and circle) is given in [16]. According to Appendix C-2 the time derivative considering only translation motion of a generic $(2D)$ i image point $(\dot{m}^{\tau}_{2D_i})$ is

$$\begin{bmatrix} \dot{x}_{2D_i} \\ \dot{y}_{2D_i} \end{bmatrix} = \begin{bmatrix} -\frac{1}{Z^C_i} & 0 & \frac{x_{2D_i}}{Z^C_i} \\ 0 & -\frac{1}{Z^C_i} & \frac{y_{2D_i}}{Z^C_i} \end{bmatrix} \begin{bmatrix} v^{C,O}_{x_C} \\ v^{C,O}_{y_C} \\ v^{C,O}_{z_C} \end{bmatrix} = L_{x_i} \mathbf{v}^{C,O}_C \tag{3-26}$$

According to Appendix 3-9 the time derivative of the difference between two generic $2D$ k and j image points along the y direction considering only translational motion is

$$\dot{d}_{y_{2D_{k,j}}} = \dot{y}_{2D_k} - \dot{y}_{2D_j} \begin{bmatrix} 0 & (-\frac{1}{Z^C_k} + \frac{1}{Z^C_j}) & (\frac{y_{2D_k}}{Z^C_k} - \frac{y_{2D_j}}{Z^C_j}) \end{bmatrix} \mathbf{v}^{C,O}_C \tag{3-27}$$

Assuming that the visual marker is planar is possible to state that all the visual marker points have the same distance from the camera. This means that $(Z^C_k = Z^C_j = Z^C_i)$. In this thesis the term (d_m) is used to indicate the distance between the planar marker and the origin of the camera frame expressed in meters. Applying the assumption that the marker is planar to both Eq. (3-26) and Eq. (3-27) is possible to state that the derivative of the estimated terms $(\hat{x}_{2D_5}, \hat{y}_{2D_5}, \hat{d}_{y_{2D_{2,3}}})$ required in the formulation of the error shown in Eq. (3-25) is

$$\begin{bmatrix} \hat{\dot{x}}_{2D_5} \\ \hat{\dot{y}}_{2D_5} \\ \hat{\dot{d}}_{y_{2D_{3,2}}} \end{bmatrix} = \begin{bmatrix} -\frac{1}{d_m} & 0 & \frac{\hat{x}_{2D_5}}{d_m} \\ 0 & -\frac{1}{d_m} & \frac{\hat{y}_{2D_5}}{d_m} \\ 0 & 0 & \frac{\hat{y}_{2D_3} - \hat{y}_{2D_2}}{d_m} \end{bmatrix} \begin{bmatrix} v^{C,O}_{x_C} \\ v^{C,O}_{y_C} \\ v^{C,O}_{z_C} \end{bmatrix} = L_x \mathbf{v}^{C,O}_C \tag{3-28}$$

The interaction matrix (L_x) is made up by the $(2D)$ image coordinates of the estimated center of the visual marker $(\hat{x}_{2D_5}$,$\hat{y}_{2D_5})$, the current distance (d_m) between the planar visual marker and the current camera frame, and the difference $(d_{y_{2D_{3,2}}} = y_{2D_3} - y_{2D_2})$ between the third and the second estimated $(2D)$ image points along the (y) direction of the $(2D)$ image coordinate system. [10] proposes other two different ways to compute the interaction matrix. The first one is to replace the measured values $(\hat{x}_{2D_5}$,$\hat{y}_{2D_5}, d_m, d_{y_{2D_{3,2}}})$ with the desired one $(x^{\star}_{2D_5}$,$y^{\star}_{2D_5}, d^{\star}_m, d^{\star}_{y_{2D_{3,2}}})$. This result in a constant interaction matrix $L_{x^{\star}}$ defined as

$$L_{x^{\star}} = \begin{bmatrix} -\frac{1}{d^{\star}_m} & 0 & \frac{x^{\star}_{2D_5}}{d^{\star}_m} \\ 0 & -\frac{1}{d^{\star}_m} & \frac{y^{\star}_{2D_5}}{d^{\star}_m} \\ 0 & 0 & \frac{y^{\star}_{2D_3} - y^{\star}_{2D_2}}{d^{\star}_m} \end{bmatrix} \tag{3-29}$$

The second approach aims at combined both the measured (L_x) and the desired $(L_{x^{\star}})$ interaction matrices to derive an average interaction matrix (L_{x_a}) defined as

$$L_{x_a} = \frac{L_x}{2} + \frac{L_{x^{\star}}}{2} = \frac{1}{2} \begin{bmatrix} -\frac{1}{d_m} - \frac{1}{d^{\star}_m} & 0 & \frac{\hat{x}_{2D_5}}{d_m} + \frac{x^{\star}_{2D_5}}{d^{\star}_m} \\ 0 & -\frac{1}{d_m} - \frac{1}{d^{\star}_m} & \frac{\hat{y}_{2D_5}}{d_m} + \frac{y^{\star}_{2D_5}}{d^{\star}_m} \\ 0 & 0 & \frac{\hat{y}_{2D_3} - \hat{y}_{2D_2}}{d_m} + \frac{y^{\star}_{2D_3} - y^{\star}_{2D_2}}{d^{\star}_m} \end{bmatrix} \tag{3-30}$$

According to the chosen interaction matrix the behavior of the controller will result not aggressive ($L_{x^\star}$), aggressive (L_x) or average (L_{x_a}). Given the derivative of a ($2D$) image coordinate provided in Appendix C-2 and the derivative of the distance among two ($2D$) image coordinate given in Appendix C-3 is possible to calculate the derivative of the error showed in Eq. (3-25) assuming that the desired ($2D$) image coordinates ($x^\star_{2D_5}, y^\star_{2D_5}, d^\star_{y_{2D_{3,2}}}$) do not move on the image plane (τ) where the current image is acquired and where the estimated ($2D$) image points move depending on the real position of the visual marker.

$$\dot{e} = \begin{bmatrix} \dot{\hat{x}}_{2D_5} - \cancel{\dot{x}^\star_{2D_5}}^{\,0} \\ \dot{\hat{y}}_{2D_5} - \cancel{\dot{y}^\star_{2D_5}}^{\,0} \\ \dot{\hat{d}}_{y_2D_{3,2}} - \cancel{\dot{d}^\star_{y_2D_{3,2}}}^{\,0} \end{bmatrix} = L_e \mathbf{v}_C^{C,O} \tag{3-31}$$

with ($L_e = L_x$) or ($L_e = L_{x^\star}$) or ($L_e = L_{x_a}$).
Given the three possible choice for the interaction error matrix the determinant of the desired ($L^\star_x$), measured (L_x) and average interaction matrix is computed. The following conclusion have been derived.

- Measured Interaction Matrix Singularity:

$$\begin{aligned} &det(L_x) = \frac{y_{2D_3} - y_{2D_2}}{d_m^3} = 0 \\ &if \quad y_{2D_3} = y_{2D_2} \quad \& \quad d_m \neq 0 \end{aligned} \tag{3-32}$$

  The condition that the estimated distance has to be different from zero ($d_m \neq 0$) it is always achieved because it is not possible to see the visual marker when the quadrotor camera is really close to it ($d_m \approx 0$). This means that to be able to see the visual marker and calculate an estimation of the distance (d_m) it is required a minimum distance between visual marker and camera allowing the visual marker to completely appear on the image plane. The second condition regards the difference along the (y) direction between the third and the second ($2D$) image points. This difference represents a measure of the side of the visual marker. The latter can approach to zero only if the marker rotate on the image plane. To avoid this problem it has been chosen to do not consider the marker orientation in calculating the markers corners. The latter are computed using the center and the side of the detected marker see Figure 3-7 and Figure 3-8. Thanks to this choice the difference ($y_{2D_3} - y_{2D_2}$) it will always be different from zero. Indeed it can only be zero or get close to zero if the visual marker is a point. However this is not the case and thanks to the detector limitation there will always be a minimum marker size with ($y_{2D_3} \neq y_{2D_2}$) at which it is possible to detect the marker. If the size will appear smaller than the minimum size the corners will not be detected anymore. Thus it is possible to conclude that the condition required to avoid that the measured interaction matrix is singular are satisfied.

- Desired Interaction Matrix Singularity;

$$\begin{aligned} &det(L^\star_x) = \frac{y^\star_{2D_3} - y^\star_{2D_2}}{d_m^{3\star}} = 0 \\ &if \quad y^\star_{2D_3} = y^\star_{2D_2} \quad \& \quad d^\star_m \neq 0 \end{aligned} \tag{3-33}$$

In this case to avoid singularity issues is required to select the desired parameters $(y^{\star}_{2D_3}, y^{\star}_{2D_2}, d^{\star}_m)$ such that the condition to avoid singularity are met. $(y^{\star}_{2D_3}, y^{\star}_{2D_2}, d^{\star}_m)$ are fixed in time and they represent the desired values at which the estimated ones have to converge. Usually in setting the mission the user will choose the desired distance between camera and marker and the center of the desired marker on the image plane. Given these values the desired corners are compute as shown in the perception module outputs 3-1-2 . The desired corners computation is equivalent at extract the desired corners from a chosen picture where the visual marker appears in the desired position. Therefore to be able to get the pixel points it is required that the marker is big enough so that its corners and center can be extracted. Therefore it will never be possible that $(y^{\star}_{2D_3} = y^{\star}_{2D_2})$ is satisfied. The latter means that the marker has turned into a point where all the marker's corners and center are the same point. In choosing properly the desired parameters it is possible to avoid singularity issues related with the desired interaction matrix.

- Average Interaction Matrix Singularity:

$$\begin{aligned} &det(L_{x_a}) = \frac{(d_m + d^{\star}_m)^2 (d_m(y^{\star}_{2D_3} - y^{\star}_{2D_2}) + d^{\star}_m(y_{2D_3} - y_{2D_2}))}{(8 d^3_m d^{\star 3}_m)} = 0 \\ &if \quad d_m(y^{\star}_{2D_3} - y^{\star}_{2D_2}) = -d^{\star}_m(y_{2D_3} - y_{2D_2}) \quad \& \quad d_m, d^{\star}_m \neq 0 \end{aligned} \tag{3-34}$$

The latter is the result of the combination of the measured and desired interaction matrix. To avoid singularity in this case both the desired and the estimated distances have to be different from zero as explained before. Then, a second condition is required which is $(d_m(y^{\star}_{2D_3} - y^{\star}_{2D_2}) \neq -d^{\star}_m(y_{2D_3} - y_{2D_2}))$. The latter is always satisfied because in dealing with no orientation of the marker the desired corners will always have the same sign of the detected ones. Furthermore also the distances $(d_m, d^{\star}_m)$ need to have the same sign which is always positive for convention. For this reason it is not possible that $(d_m(y^{\star}_{2D_3} - y^{\star}_{2D_2}) = -d^{\star}_m(y_{2D_3} - y_{2D_2}))$ because this means that the desired and estimated values will have opposite signs. It is like having the quadrotor approaching the visual marker upside down that is not feasible. A situation in which the condition can be satisfied is when the quadrotor's bottom camera is used to land on platform where a visual marker attached to it is rotating about the vertical axis of the platform. This situation can cause the desired and the estimated values to be equal and with different sign when the platform is rotated 180 degree with respect to the desired values. However, even if this occurs it has been chosen to extract the pixel points always assuming that the visual marker orientation does not change and for this reason the condition $(d_m(y^{\star}_{2D_3} - y^{\star}_{2D_2}) \neq -d^{\star}_m(y_{2D_3} - y_{2D_2}))$ is always satisfied.

The singularity of the error interaction matrix (L_e) has been studied and it is possible to conclude that there are not singularity issues related with the desired, measured and average interaction matrix.

(c) **Compute $\mathbf{v}^{\mathbf{C,O}}_{\mathbf{C}}$**
Imposing an exponential error decay $(\dot{e} = -\lambda e)$ is possible to retrieve the camera

instantaneous translational camera velocities

$$\mathbf{v}_C^{C,O} = -\lambda \hat{L}_e^{-1} e \rightarrow \begin{bmatrix} v_{x_C}^{C,O} \\ v_{y_C}^{C,O} \\ v_{z_C}^{C,O} \end{bmatrix} = -\lambda \hat{L}_e^{-1} \begin{bmatrix} \hat{x}_{2D_5} - x_{2D_5}^{\star} \\ \hat{y}_{2D_5} - y_{2D_5}^{\star} \\ \hat{d}_{y_{2D_{3,2}}} - d_{y_{2D_{3,2}}}^{\star} \end{bmatrix} \tag{3-35}$$

where (λ) is a positive convergence factor. ($\mathbf{v}_C^{C,O}$) represents the translational velocities of the camera frame (C) with respect to the visual marker frame (O) expressed in camera coordinate frame (C) . The goal of this section is to map these velocities into the input velocities of the navigation controller framework which are the velocities of the quadrotor robot frame (R) with respect to the visual marker frame (O) expressed in world coordinate frame ($\mathbf{v}_R^{W,O}$). The rotation matrices describing how the front and bottom camera frame are rotated with respect to the robot frame have been presented in 3-1-2. An illustration showing how the front camera, bottom camera, robot and world frames are oriented one respect to each other is presented in Figure 3-4. Given these values is possible to derive the velocity of the camera frame (C) with respect to the visual marker frame (O) expressed in robot coordinate frame (R) as follows

$$\mathbf{v}_C^{R,O} = R_C^R \mathbf{v}_C^{C,O} \rightarrow \begin{bmatrix} v_{x_C}^{R,O} \\ v_{y_C}^{R,O} \\ v_{z_C}^{R,O} \end{bmatrix} = R_C^R \begin{bmatrix} v_{x_C}^{C,O} \\ v_{y_C}^{C,O} \\ v_{z_C}^{C,O} \end{bmatrix} \tag{3-36}$$

with ($R_C^R = R_{C_f}^R$) if the chosen camera is the front one or ($R_C^R = R_{C_b}^R$) if the selected camera is the bottom one. The rotation matrices values ($R_{C_f}^R, R_{C_b}^R$) are given in Eq. (3-1). Knowing that the chosen camera is fixed with the quadrotor and assuming that both the camera's frames have their origin on the robot frame it is possible to state that the velocities of the camera frame (C) with respect to visual marker (O) expressed in robot coordinate frame (R) is equivalent to the velocities of the robot frame (R) with respect to visual marker (O) expressed in robot coordinate frame (R). This assumption is possible because the two cameras are fixed with the quadrotor itself which means that moving the quadrotor the camera will move as well in the same way. Thus it is possible to infer

$$\mathbf{v}_R^{R,O} = \mathbf{v}_C^{R,O} \tag{3-37}$$

Given ($\mathbf{v}_R^{R,O}$) is required to map these velocities into the corresponding velocities expressed in world coordinate frame. To do this the rotation matrix describing how the robot frame (R) is rotated with respect to the World frame (R_R^W) is required. The latter has been present in B-1-1 and its value is given in Eq. (B-2). Once the rotation matrix ($R_R^W(\psi, \theta, \phi)$) is available the translational velocities of the robot frame (R) with respect to the visual marker (O) expressed in world coordinate frame (W) are given by

$$\mathbf{v}_R^{W,O} = R_R^W \mathbf{v}_R^{R,O} \rightarrow \begin{bmatrix} v_{x_R}^{W,O} \\ v_{y_R}^{W,O} \\ v_{z_R}^{W,O} \end{bmatrix} = R_R^W \begin{bmatrix} v_{x_R}^{R,O} \\ v_{y_R}^{R,O} \\ v_{z_R}^{R,O} \end{bmatrix} \tag{3-38}$$

3-3-3 Algorithm

A summary of the overall algorithm running inside the image based visual servo module is provided.

Algorithm 11 Image based visual servo controller module

Initialization:

Get default $(safety\_counter\_threshold, Min, Max)$
Get default variables front camera:
${}^f\lambda = diag({}^f\lambda_{11}, {}^f\lambda_{22}, {}^f\lambda_{33}), {}^f interaction\_matrix$
Get default variables bottom camera:
${}^b\lambda = diag({}^b\lambda_{11}, {}^b\lambda_{22}, {}^b\lambda_{33}), {}^b interaction\_matrix$

Algorithm: (Run at same frequency of image state estimator module)

Get $chosen\_camera$
Get $safety\_counter$
Get R_C^R and R_R^W
Get $(\hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}, m^{\tau^\star}_{2D_{i=1,2,\cdots 5}}, d^\star_m, d_m)$
if chosen_camera $= 0$ **then**
  Set $(\lambda, interaction\_matrix) = ({}^b\lambda, {}^b interaction\_matrix)$
else
  Set $(\lambda, interaction\_matrix) = ({}^f\lambda, {}^f interaction\_matrix)$
end if
Compute $e = \mathbf{f}(\hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}, m^{\tau^\star}_{2D_{i=1,2,\cdots 5}})$
if $interaction\_matrix = 0$ (constant interaction matrix) **then**
  Compute $L_e = \mathbf{f}(m^{\tau^\star}_{2D_{i=1,2,\cdots 5}}, d^\star_m)$
else if $interaction\_matrix = 1$ (measured interation matrix) **then**
  Compute $L_e = \mathbf{f}(\hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}, d_m)$
else if $interaction\_matrix = 2$ (average interaction matrix) **then**
  Compute $L_e = \mathbf{f}(m^{\tau^\star}_{2D_{i=1,2,\cdots 5}}, \hat{m}^{\tau}_{2D_{i=1,2,\cdots 5}}, d^\star_m, d_m)$
else
  Compute $L_e = \mathbf{f}(m^{\tau^\star}_{2D_{i=1,2,\cdots 5}}, d^\star_m)$
end if
Compute $\mathbf{v}_C^{C,O} = -\lambda L_e^{-1} e$
Compute $\mathbf{v}_C^{W,O} = R_R^W R_C^R \mathbf{v}_C^{C,O}$ (Assumption: camera frame fixed with the quadrotor) $\rightarrow \mathbf{v}_R^{W,O} = \mathbf{v}_C^{W,O}$
Check if $\mathbf{v}_R^{W,O} \in \begin{bmatrix} Min & Max \end{bmatrix}$, if NOT $\rightarrow$ saturate
if $safety\_counter > safety\_counter\_threshold$ **then**
  $\mathbf{v}_R^{W,O} = 0_{3\times 1}$ (hover)
end if
Send $\mathbf{v}_R^{W,O}$ (Navigation controller input)

A resume of inputs and outputs of the image bases visual servo controller module is provided.

$$\left[\begin{array}{c}\textbf{INPUTS}\\ 1)\underline{\textbf{chosen\_camera}} : (0 = bottom, 1 = front)\\ 2)\underline{\textbf{safety\_counter}}\\ 3)\underline{\textbf{safety\_counter\_threshold}}\\ 4)Default\ variables\ front\ camera :\\ \underline{^{\mathbf{f}}\lambda, {}^{\mathbf{f}}\textbf{interaction\_matrix}}\\ 5)Default\ variables\ bottom\ camera :\\ \underline{^{\mathbf{b}}\lambda, {}^{\mathbf{b}}\textbf{nteraction\_matrix}}\\ 6)Fixed\ rotation\ matrix\ from\\ camera\ frame\ to\ robot\ frame : \underline{\mathbf{R}_{\mathbf{C}}^{\mathbf{R}}}\\ 7)NOT\ fixed\ rotation\ matrix\ from\\ robot\ frame\ to\ world\ frame : \underline{\mathbf{R}_{\mathbf{R}}^{\mathbf{W}}}\\ 9)Estimated\ pixel\ coordinates : \underline{(\hat{\mathbf{m}}_{\mathbf{2D_{i=1,2\cdots 5}}}^{\tau})}\\ 9)Desired\ pixel\ coordinates : \underline{(\mathbf{m}_{\mathbf{2D_{i=1,2,\cdots 5}}}^{\tau^\star})}\\ 10)Estimated\ distance\ in\ meters\ between\\ chosen\ camera\ frame\ and\ marker\\ (Assumption\ Marker\ is\ planar) : \underline{\mathbf{d_m}}\\ 11)Desired\ distance\ in\ meters\ between\\ chosen\ camera\ frame\ and\ marker\\ (Assumption\ Marker\ is\ planar) : \underline{\mathbf{d}_{\mathbf{m}}^{\star}}\\ 12)Saturation\ IBVS\ output\ velocities\\ constraints : \underline{(\mathbf{Min}, \mathbf{Max}}\end{array}\right] \rightarrow \left[\begin{array}{c}\textbf{OUTPUTS}\\ 1)Velocities\ of\ the\ robot\\ with\ respect\ to\ the\ marker\\ expressed\ in\ world\ coordinate\ frame :\\ \underline{\mathbf{v}_{\mathbf{R}}^{\mathbf{W},\mathbf{O}}}\end{array}\right]$$

Chapter 4

Experiment

4-1 Experiment description

The objective of this chapter is to validate how the vision based planner and the navigation controller framework perform in solving a fully autonomous chosen mission. The mission chosen to validate both the vision based planner and the navigation controller framework designed is the following.

Mission

1. *Take_off*
2. *Go_to_point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 0, 0, 1.3, 0 \end{bmatrix}$
3. *Go_to_point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 0, 2, 1.3, 0 \end{bmatrix}$
4. *Rotate* $\rightarrow \psi^{\star W} = -90$
5. *Go_to_point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 1.5, 2, 1.3, -90 \end{bmatrix}$
6. *Approach_to_an_object* $\rightarrow$ *camera = 1 (front camera)*
7. *Go_to_point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 1.5, 2, 1.3, -90 \end{bmatrix}$
8. *Rotate* $\rightarrow \psi^{\star W} = -180$
9. *Go_to_point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 1.5, 0, 1.3, -180 \end{bmatrix}$
10. *Rotate* $\rightarrow \psi^{\star W} = 90$
11. *Go_to_point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 0, 0, 1.3, 90 \end{bmatrix}$
12. *Rotate* $\rightarrow \psi^{\star W} = 0$
13. *Go_to_point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 0, 0.5, 3, 0 \end{bmatrix}$
14. *Approach_to_an_object* $\rightarrow$ *camera = 0 (bottom camera)*
15. *Land*

The upper mission has been designed as a sequence of task taking different inputs parameters that can be used multiple times in the mission to achieve different goals simply assigning to the task input parameters with different values. In designing the mission the following quadrotor tasks have been chosen.

- ***Go_to_point*** $(x^{\star W}, y^{\star W}, z^{\star W}, \psi^{\star W}) \rightarrow$ ***Reach a desired pose***: The goal of this task it is given a desired pose $(x^{\star W}, y^{\star W}, z^{\star W}, \psi^{\star W})$ to drive the quadrotor from it its current pose up to the desired one. This is accomplished using the navigation controller framework and a state estimator in charge of estimating the quadrotor full states (position, velocity, acceleration, orientation).
- ***Rotate*** $(\psi^{\star W}) \rightarrow$ **Rotate keeping the same position**: This task has the objective to make the quadrotor rotate of a desired yaw angle $(\psi^{\star W})$ minimizing the quadrotor's displacement from its current position. This task is also solved combining the navigation controller framework with the state estimator. In this task it is not required to specify the desired quadrotor position as a input parameter.
- ***Approach_to_an_object*** $(camera = 1) \rightarrow$ **Approach to a known either static**

or moving ArUco Marker up to a desired distance: This task has the aim to make a quadrotor able to approach to either a static or moving ArUco marker up to a desired distance from it keeping the center of the detected ArUco marker on a desired position on the acquired camera image. To solve this task it has been chosen to use the quadrotor front camera and to keep the ArUco marker static. This task is achieved combining the vision based planner with the navigation controller framework. A state estimator is also required with the aim to provide velocities and yaw orientation quadrotor's measurements. In dealing with this task the quadrotor does not require to know neither its position in space nor the position of the marker. Indeed, it is sufficient that the visual marker appears on the acquired camera image.

- ***Approach_to_an_object*** $(camera = 0)$ + ***Land*** $\rightarrow$ **Autonomously land on a static platform.** This task can be split in two subtasks. The first one is to approach a platform on which an ArUco marker is placed up to a desired distance from it. It has been chosen to keep the platform static. In doing this task it has been chosen to keep the platform static. The second one it is to land on this platform when the quadrotor has reached the desired distance. Thus this task has been solved combining the vision based planner with the navigation controller framework using the quadrotor bottom camera. Also in this case a state estimator is required with the aim to provide velocities and yaw orientation quadrotor's measurements .

The mission will be performed in both real flight using the AR Drone 2.0 and in simulation environment using the Gazebo simulator in combination with the PX4 Software-In-The-Loop. The specification of the two choices are provided.

- **Real flight mission (AR Drone 2.0)**: To perform the mission in real flight an AR drone 2.0 has been chosen. The specification of the latter are given in Appendix A whereas the control technology allowing it to take off, land and hover without drifting is presented in [17]. In solving the real flight mission the developed navigation controller framework and vision based planner are combined with the Aerostack EKF state estimator and with the mission planner. Furthermore the driver package *ardrone_autonomy* it is used to receive the onboard sensors data measurements and to send autopilot commands allowing to control as desired the quadrotor. In dealing with the AR Drone 2.0 the *Take_off* and the *Land* tasks described in 4-1 are performed simply sending to the *ardrone_autonomy* ROS driver a specific command for the taking off task and another for the landing task.

- **Simulated flight mission (Gazebo + PX4 SITL)**: For the simulation flight it has been chosen to use the Gazebo simulator combined with the PX4 Software-In-The-Loop (SITL) [18]. This simulation runs the same firmware that is running on a real Pixahwk[1] autopilot. Therefore to test how the navigation controller framework can be combined with the Pixhawk autopilot it has been preferred to use the simulation environment to avoid useless crashes. In this mission the gazebo ground truth data have been used to accurately estimated the quadrotor states in the simulation environment. In making this choice it is possible to forget the state estimation problems and validate only the vision based planner and the navigation controller framework in an almost ideal

[1] https://pixhawk.org/modules/pixhawk

condition. The Aerostack mission planner has been used as well in combination with the developed vision based planner and navigation controller framework to make the quadrotor performs the desired tasks. The ROS driver *mavros* package has been used for both receiving data from the Pixhawk autopilot and for sending autopilot commands to the latter. The *Take_off* and the *Land* tasks in dealing with a quadrotor equipped with Pixhawk autopilot have to be designed . Indeed there are not commands such as in the AR Drone 2.0 to ensure a stable take off and land. To overcome this problem the *Take_off* task has been designed as a *Go_to_point* task which means that to do solve that both the navigation controller framework and the ground truth measurements are required. About the *Land* task it has been chosen to design it simply shutting down the motor.

To be able to combine the upper tasks in a unique fully autonomous mission the Aerostack mission planner has been used. The idea is that each task has to be able to be used multiple times in a mission according to a different set of parameters that are used to configure the mission itself. For example the task *Go_to_point* can be used multiple times during the mission to make the quadrotor reach different poses. The same consideration can be done for the *Rotate* task which can be used to make the quadrotor rotate up to a desired yaw angle. Also in this case it is possible that this task can be repeated different times in the same mission. To conclude the quadrotor experiment goal is to solve a fully autonomous mission in both real and simulation flight respectively using the AR Drone 2.0 and the Gazebo simulator combined with the PX4-Software-In-The-Loop. The mission is the result of different tasks (*Take_off*, *Go_to_point*,*Rotate*,*Approach_to_an_object*, *Land*) which can be combined as desired and used multiple times with different input parameters in the same mission.

4-2 Experiment requirements

ROS has been chosen as a software framework to ensure communication among different modules. An overview of how ROS works is given in [19] whereas a more detailed explanation is available in this book [20]. The navigation controller is made up by five different modules (horizontal position controller, vertical position controller, horizontal speed controller, vertical speed controller, yaw controller) representing five different ROS packages which can run at a different frequency. The vision based planner is composed by three different modules (perception, image state estimator, image based visual servo). All the modules have been designed such that it is possible with a ROS service to start or stop them. However to be able to design the desired fully autonomous mission (4-1)‘ the vision based planner and the navigation controller framework are not enough. Indeed firstly a state estimator is requried to get an estimate of the quadrotor states (position, velocity, acceleration and orientation), secondly a mission planner is needed to be able to set up the mission and in particular to allow the quadrotor to perform different tasks consecutively, finally it is required to know how it is possible to control both the AR Drone 2.0 and a quadrotor equipped with Pixhawk autopilot using the the navigation controller framework.

4-2-1 State estimator

To make the quadrotor be able to move in space a quadrotor state estimator is required. The latter is in charge of estimating the quadrotor full states (position, velocity, acceleration, orientation) fusing together the quadrotor onboard sensors. Different choices have been done related to the state estimator for the simulation and the real flight experiment.

- **State estimator simulation experiment**: For the simulation experiment it has been preferred to use the Gazebo ground truth data measurements to test how the vision based planner and the navigation controller framework perform with a quadrotor equipped with a Pixhawk autopilot. Indeed the PX4 Software-In-The-Loop has been used. It has been preferred to use the ground data in this experiment to do not add further estimation problems in the validation of both the vision based planner and the navigation controller framework. In making this choice it is possible to focus only on how the developed modules behave in an almost ideal condition. Only when the simulation has been validated it is possible to move to a real flight scenario in which an EKF is used to fuse the different on board sensors to provide the quadrotor full states estimation.

- **State estimator real flight experiment**: In dealing with real flight performed with the AR Drone 2.0 the estimation of the quadrotor states has been provided by an EKF module available in the Aerostack. The latter fuses together the IMU $(\theta, \phi, \psi, \dot{\theta}, \dot{\phi}, \dot{\psi}, a_x, a_y, a_z)$, the optical flow $(\dot{x}, \dot{y})$ and the ultrasound altimeter $(z, \dot{z})$ data provided by the ROS driver *ardrone_autonomy* package to obtain an estimation of the quadrotor states $(\hat{x}^W, \hat{y}^W, \hat{z}^W, \hat{\dot{x}}^W, \hat{\dot{y}}^W, \hat{\dot{z}}^W, \hat{\phi}^W, \hat{\theta}^W, \hat{\psi}^W)$ expressed in world coordinate frame following ENU convention. However the estimation of the position is performed integrating the velocities and for this reason the quadrotor will accumulated a drift in the estimation of the latter that it will increase in time. To validate the vision based planner and the navigation controller framework it has been chosen to use the estimated EKF data rather than the ground truth data provided by a MOCAP system. This choice is related to the fact that the quadrotor velocities are estimated correctly by the EKF. This means that in making the quadrotor approaching an object up to a desired distance all the required estimated data are available using the EKF. However in trying to drive the quadrotor from the current to a chosen desired pose the position estimation will accumulated drift in time. Anyway having a wrong position estimation does not affect how the navigation controller framework tracks the desired reference. If the measurements are wrong the navigation controller framework will track the wrong ones. The drawback is that it thinks to have reached a pose in space that it is not the real one. Furthermore, it is has been chosen to do not rely on the MOCAP system in doing the mission because it is an expensive system that cannot be moved freely from one location to another. For this reason it has be preferred to validate the developed modules using EKF estimated data rather than the ground truth ones to increase the level of autonomy of the mission. Anyway the ground truth data have been collected for comparison purpose. The EKF estimated data are provided at a frequency of $(30Hz)$ because optical flow data are derived from sequences of images acquired by the bottom camera of the AR Drone which runs at $(30Hz)$. The estimated data are used as feedbacks in the navigation controller framework (see Figure 2-1).

4-2-2 Mission planner

They are a set of modules available in the Aerostack that allow to define a mission in which the quadrotor has to face different tasks. Each quadrotor task is solved thanks to the combination of different modules that has to be started. When the task is solved all the modules associated with the solved task are stopped and a new task with new associated modules can start. To be able to perform different tasks in the same mission a mission planner is required. The Aerostack mission planner has been used to build the desired mission. The latter is composed by a python mission interpreter in charge of reading a python mission file and by an Executive system which is in charge of starting and stopping different ROS modules according to a specific task that the quadrotor has to face. A detailed explanation of how it works it available here [21],[22] [23]. The main idea behind this system is that a quadrotor mission can be divided into multiple tasks that the quadrotor has to accomplish. These tasks are called behaviors. To solve these problems the quadrotor requires different modules to be started. The knowledge of which module is associated to a certain behavior is provided a priori by an operator. To conclude, a mission is sequence of behaviors where each behavior requires a set of modules to be activated. A behavior condition is introduced inside each behavior and it is used to tell to the mission planner when a specific task is solved. Indeed, when the condition is satisfied a new behavior can start. The advantage of this approach it is that simply modifying the behavior parameters and arranging them in a different order it is possible to build different mission in a fast an easy way. For the sake of this thesis three different behaviors have been used to design the final mission used to validate the vision based planner and the navigation controller framework. They are summarized as follows

- *Go_to_point*: This behavior takes as input the desired pose $(x^{\star W}, y^{\star W}, z^{\star W}, \psi^{\star W})$ from the mission file. Then, it activates all the modules associated to it. The modules associated to this behavior in both the real flight and in the simulated flight are provided in the following table.

Table 4-1: The table shows the *Go_to_point* behavior's inputs and the modules that has to be started to solve the *Go_to_point* task in both real and simulation flight.

Behaviors	Required ROS packages AR Drone 2.0	Required ROS packages Gazebo + PX4 SITL
Go_to_point Inputs: $(x^{\star W}, y^{\star W}, z^{\star W}, \psi^{\star W})$	horizontal position controller vertical position controller horizontal speed controller yaw controller EKF	ground truth horizontal position controller vertical position controller horizontal speed controller vertical speed controller yaw controller

This means that to be able to perform the two missions it is required to distinguish among modules associated to the behavior *Go_to_point* in the real and in the simulation flight. Given Table 4-1 it is possible to state that the behavior *Go_to_point* goal is to drive a quadrotor from it is current pose to a desired one. The choice of the desired pose is provided in the mission file. Because a behavior represents a quadrotor task a condition is required to end the task. The stop condition associated to the behavior

Go_to_point is

$$\sqrt{(x^{\star W} - \hat{x}^W)^2 + (y^{\star W} - \hat{y}^W)^2 + (z^{\star W} - \hat{z}^W)^2 + (\psi^{\star W} - \hat{\psi}^W)^2} < \sigma_p \tag{4-1}$$

which means that the quadrotor has reached the desired pose with a pose error smaller than a chosen preloaded threshold value (σ_p). The latter is used to increase or decrease the accuracy of the behavior *Go_to_point*.

- *Rotate*: This behavior takes as input the desired yaw angle ($\psi^{\star W}$) from the mission file. Then, it activates all the modules associated to it. The modules associated to this behavior in both the real flight and in the simulated flight are provided in the following table.

Table 4-2: The table shows the *Rotate* behavior's input and the modules that has to be started to solve the *Rotate* task in both real and simulation flight.

Behaviors	Required ROS packages AR Drone 2.0	Required ROS packages Gazebo + PX4 SITL
Rotate Input: ($\psi^{\star W}$)	horizontal position controller vertical position controller horizontal speed controller yaw controller EKF	ground truth horizontal position controller vertical position controller horizontal speed controller vertical speed controller yaw controller

According to Table 4-2 it is possible to see that the same modules of the *Go_to_point* behavior are started. The difference between the two modules it is that the *Rotate* behavior only takes as input the desired yaw angle an not the position that the quadrotor has to reach. This means that this behavior allows the quadrotor to rotate on it is current position. Indeed the desired position in this case it is the first position measurements that the state estimator provides when the *Rotate* behavior is started. The stop condition associated to the behavior *Rotate* is

$$|\psi^W - \hat{\psi}^W| < \sigma_\psi \tag{4-2}$$

which means that the quadrotor has reached the desired yaw angle with an error smaller than a chosen preloaded threshold value (σ_ψ). The latter is used to increase or decrease the accuracy of the behavior *Rotate*. It has been preferred to start all the controllers of the navigation controller framework and not only the yaw controller because it is better if the quadrotor rotate keeping itself in the same position. To do this all the navigation controller framework are required.

- *Approach_to_an_object*: This behavior takes as input the desired camera that it has to use to approach an object. This input parameter is used to tell to the vision based planner which camera it has to use either front or bottom for the solution of the task. Simply changing this value the *Approach_to_an_object* behavior can be used to follow an object using the front camera of to autonomously land either on a static or moving platform using the bottom camera. The modules associate to this behavior in both the real flight and in the simulated flight are provided in the following table.

Table 4-3: The table shows the behavior used in the chosen mission together with the module associated to them for both real and simulated flight.

Behaviors	Required ROS packages AR Drone 2.0	Required ROS packages Gazebo + PX4 SITL
Approach_to_an_object Input: (*chosen_camera*)	perception image state estimator image based visual servo horizontal speed controller yaw controller EKF	ground truth perception image state estimator image based visual servo horizontal speed controller vertical speed controller yaw controller

From Table 4-3 it is possible to see that this behavior combines both the developed vision based planner and the navigation controller framework to solve the task. The condition chosen to stop the behavior is that the IBVS error defined in Eq. (3-25) is smaller than a chosen threshold value (σ_{IBVS}). This behavior is in charge of starting all the modules required to make the quadrotor approach to an object up to a desired distance keeping the center of the latter in a precise desired image position. Thus it is possible to conclude that the condition used to stop the modules is

$$e_{IBVS} < \sigma_{IBVS} \tag{4-3}$$

To conclude it is possible to say that the design of the mission presented in 4-1 is a combination of behaviors which requires different modules to be activated. The main advantage of this design it is that the same behavior can be used multiple times in the same mission and it can be configured with different input parameters. In addition combining the presented behaviors in a different way it is possible to easily design different missions.

4-2-3 Autopilot drivers

To be able to communicate with AR Drone 2.0 and with the Pixhawk autopilot two different open source ROS drivers are required.

- *ardrone_autonomy* [2]: This ROS package it is used to control the AR. Drone 2.0 using ROS as a software communication framework. Thanks to this package it is possible to both read the AR Drone 2.0 onboard sensor data and send Parrot autopilot commands $({}^{PA}\theta^{\star}, {}^{PA}\phi^{\star}, {}^{PA}\dot{z}^{\star}, {}^{PA}\dot{\psi}^{\star})$ to control as desired the quadrotor.

- *mavros*[3]: This ROS package it is used to coomunicate with the Pixhawk autopilot. In particular the mavros package it is used to deal with a MAVLink communication protocol were MAV stands for Micro Aerial Vehicle. Thanks to this ROS package it is possible to both read the Pixhawk autopilot sensor data and send Pixahwk autopilot commands $({}^{PI}\theta^{\star R}, {}^{PI}\phi^{\star R}, {}^{PI}\psi^{\star R}, {}^{PI}T^{\star R})$ to control the quadrotor.

[2] http://wiki.ros.org/ardrone_autonomy
[3] http://wiki.ros.org/mavros

However different autopilots requires different autopilot commands to be sent and different convention can be used to express them. For this reason it is required to find the relation between the navigation controller framework outputs and both the Parrot and Pixahwk autopilot commands. In the following subsection it is explained how the navigation controller framework outputs are related with both the Parrot and Pixhawk ones. (see Figure 2-1).

4-2-4 How to use the navigation controller framework to control an AR. Drone 2.0?

The Parrot AR Drone 2.0 autopilot commands $({}^{PA}\theta^{\star}, {}^{PA}\phi^{\star}, {}^{PA}\dot{z}^{\star}, {}^{PA}\dot{\psi}^{\star})$ convention used in the ROS *ardrone_autonomy* driver package is the following.

$$\begin{aligned} {}^{PA}\theta^{\star R} &\in \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (+forward, -backward) \\ {}^{PA}\phi^{\star R} &\in \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (+leftward, -rightward) \\ {}^{PA}\dot{z}^{\star R} &\in \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (+upward, -downward) \\ {}^{PA}\dot{\psi}^{\star R} &\in \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (+ccw, -cw) \end{aligned} \tag{4-4}$$

Given the navigation controller framework conventions available on Table 2-1 and illustrated on Figure 2-2 the mapping between Parrot Autopilot commands $({}^{PA}\theta^{\star R}, {}^{PA}\phi^{\star R}, {}^{PA}\dot{z}^{\star R}, {}^{PA}\dot{\psi}^{\star R})$ and navigation controller framework commands $(\theta^{\star R}, \phi^{\star}, {}^{A}\dot{z}^{\star R}, {}^{A}\dot{\psi}^{\star R})$ is given by

$$\begin{aligned} {}^{PA}\theta^{\star R} &= \theta^{\star R} \\ {}^{PA}\phi^{\star R} &= -\phi^{\star R} \\ {}^{PA}\dot{z}^{\star R} &= \dot{z}^{\star R} \\ {}^{PA}\dot{\psi}^{\star R} &= \dot{\psi}^{\star R} \end{aligned} \tag{4-5}$$

where $({}^{PA}\theta^{\star R}, {}^{PA}\phi^{\star R}, {}^{PA}\dot{z}^{\star R}, {}^{PA}\dot{\psi}^{\star R})$ represent the Autopilot commands that is required to send to the ROS Driver package *ardrone_autonomy* to control the Parrot AR Drone 2.0. An illustration of the convention used to represent both the Parrot Autopilot and the navigation framework commands is provided.

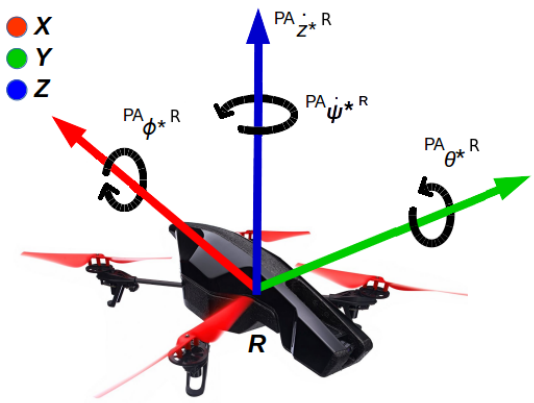


Figure 4-1: Frame convention used to express the Parrot AR Drone 2.0 Autopilots commands

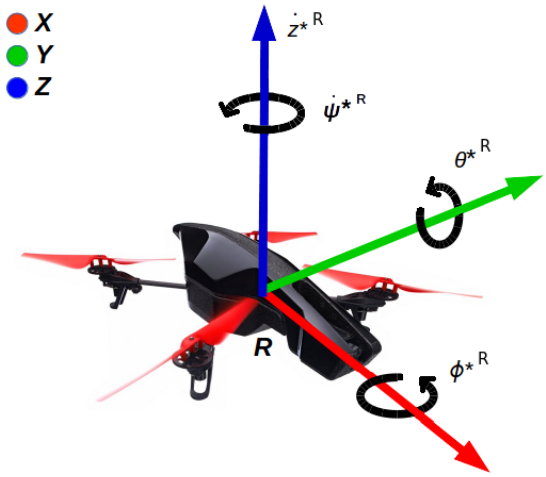


Figure 4-2: Frame convention (ENU) used for the Navigation controller framework outputs commands

It is interesting to underline that the ROS Driver package *bebop_autonomy* [4] used to control the Parrot Bebop 2.0 follows the same Autopilot commands convention of the AR Drone 2.0. Thus it is possible to infer that the autopilot commands showed in Eq. (4-5) can also be used to control the Parrot Bebop 2.0. The specification of both AR Drone 2.0 and Bebop 2.0 are given in Appendix A.

4-2-5 How to use the navigation controller framework to control a quadrotor equipped with Pixhawk autopilot?

To test the navigation controller framework with the Pixahawk autopilot [24] the Gazebo simulator combined with the PX4 Software-In-The-Loop has been used. This simulation runs the same firmware that is running on a real Pixahwk autopilot. Therefore to test how the navigation controller framework can be combined with the Pixhawk autopilot it has been preferred to use the simulation environment to avoid useless crashes. The convention used to send Pixhawk autopilots commands to the ROS driver package *mavoros* are summarized as follows

$$\begin{aligned} {}^{Pi}\theta^{\star R} &\in \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (+backward, -forward) \\ {}^{PI}\phi^{\star R} &\in \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (+rightward, -leftward) \\ {}^{PI}T^{\star R} &\in \begin{bmatrix} 0 & 1 \end{bmatrix} \quad (+upward, -downward) \\ {}^{PI}\psi^{\star R} &\in \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (+cw, -ccw) \end{aligned} \tag{4-6}$$

Given Eq. (4-6) it is possible to conclude that $({}^{PI}\theta^{\star R}, {}^{PI}\phi^{\star R}, {}^{PI}\psi^{\star R})$ follow NED convention whereas $({}^{PI}T^{\star R})$ follows ENU convention. The mapping between navigation controller out-

[4] http://wiki.ros.org/bebop_autonomy

puts $(\theta^{\star R}, \phi^{\star R}, T^{\star R}, \dot{\psi}^{\star R})$ and the autopilot commands $({}^{PI}\theta^{\star R}, {}^{PI}\phi^{\star R}, {}^{PI}T^{\star R}, {}^{PI}\psi^{\star R})$ that it is required to send to the ROS driver package *mavros* is provided.

$$\begin{aligned} {}^{PI}\theta^{\star R} &= -\theta^{\star R} \\ {}^{PI}\phi^{\star R} &= \phi^{\star R} \\ {}^{PI}T^{\star} &= T^{\star R} \\ {}^{PI}\psi^{\star} &= -\left(\psi^{R}_{IMU} + \dot{\psi}^{\star R}\right) \end{aligned} \tag{4-7}$$

where $({}^{PI}\psi^{\star R} \in \begin{bmatrix} -\pi & \pi \end{bmatrix} \begin{bmatrix} rad \end{bmatrix})$ is the reference yaw expressed in NED convention, $(\psi^{R}_{IMU} \in \begin{bmatrix} -\pi & \pi \end{bmatrix} \begin{bmatrix} rad \end{bmatrix})$ is the yaw derived from the IMU data expressed in ENU convention obtained subscribing to the */mavros/imu/data* topic and $(\dot{\psi}^{\star R} \in \begin{bmatrix} -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} rad/s \end{bmatrix})$ is the output of the yaw controller module which follows ENU convention. A sample code is available in Appendix B-3 with aim to show how it is possible to send both open loop $({}^{PI}\theta^{\star R}, {}^{PI}\phi^{\star R}, {}^{PI}T^{\star R}, {}^{PI}\psi^{\star R})$ and navigation controller outputs $(\theta^{\star R}, \phi^{\star R}, T^{\star R}, \dot{\psi}^{\star R})$ commands to the *mavros* ROS driver package to control a quadrotor equipped with Pixhawk autopilot. An illustration showing the Pixhawk autopilot commands convention with respect to the navigation controller framework ones is provided.

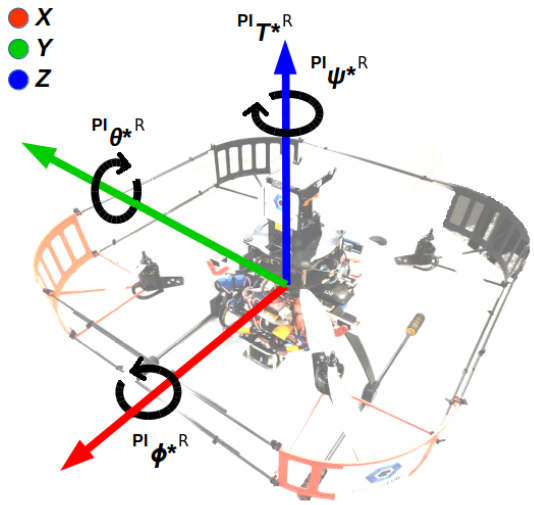


Figure 4-3: Frame convention used for the Pixhawk autopilot commands

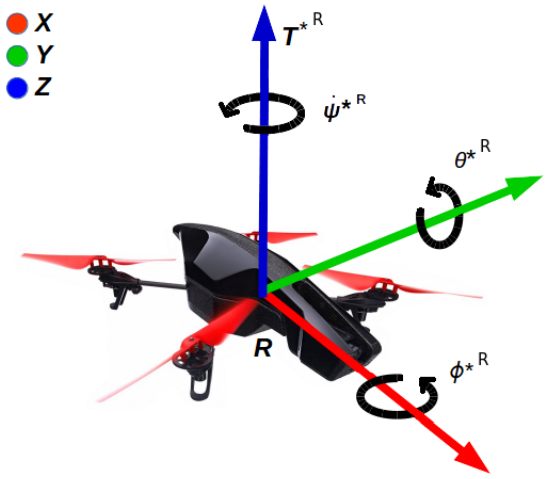


Figure 4-4: Frame convention used for the navigation controller outputs

Chapter 5

Results

In this chapter the results of the experiment described in chapter 4 are presented. In particular these results are related to the mission described in 4-1 which it has been chosen to perform in both real flight using the AR Drone 2.0 and in simulation flight combining the Gazebo simulator with the PX4 Software-In-The-Loop.

5-1 Simulation experiment

The video associated to the simulation experiment presented in this section is available here.

https://www.youtube.com/watch?v=ca6HT5rsF5Q

Another video representing the same experiment where the platform it is performing a circular motion it provided at this link (https://www.youtube.com/watch?v=IP0spKf_1Rw). For the sake of this thesis it has been preferred to see how the vision based planned perform using static platform rather than a moving one to be able to visualize clearly the trajectory of the estimated marker's corners and center with respect to the desired ones.

5-1-1 Default parameters used in the simulation experiment (Simulation Pixhawk autopilot)

The navigation controller framework and the vision based planner parameters that have been used to perform the fully autonomous mission described in 4-1 are provided. The description of their meaning together with the explanation of how they are derived is given in chapter 2 and 3.

Table 5-1: Horizontal position controller module default parameters (Simulation Pixhawk autopilot)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{hpc} = 0.1s$	$K_p^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 1$ $K_d^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 0.5$ $K_i^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 0$ $b^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 1$ $c^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 0$ $K_p^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 1$ $K_d^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 0.5$ $K_i^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 0$ $b^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 1$ $c^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 0$	$U_{Min}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = -1$ $U_{Max}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 1$ $U_{Min}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = -1$ $U_{Max}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 1$	$enable_{der_{filter}}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = True$ $N_d^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 17.3$ $enable_{ref_{filter}}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = False$ $N_r^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = N.R.$ $enable_{der_{filter}}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = True$ $N_d^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 17.3$ $enable_{ref_{filter}}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = False$ $N_r^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = N.R.$	$enable_{anti_{windup}}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = False$ $K_{aw}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = N.R.$ $enable_{anti_{windup}}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = False$ $K_{aw}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = N.R.$

Table 5-2: Vertical position controller module default parameters (Simulation Pixhawk autopilot)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{vpc} = 0.1s$	$K_p^{(z^\star,\hat{z})\rightarrow\dot{z}^\star} = 1$ $K_d^{(z^\star,\hat{z})\rightarrow\dot{z}^\star} = 0$ $K_i^{(z^\star,\hat{z})\rightarrow\dot{z}^\star} = 0$ $b^{(z^\star,\hat{z})\rightarrow\dot{z}^\star} = 1$ $c^{(z^\star,\hat{z})\rightarrow\dot{z}^\star} = 0$	$U_{Min}^{(z^\star,\hat{z})\rightarrow\dot{z}^\star} = -1$ $U_{Max}^{(z^\star,\hat{z})\rightarrow\dot{z}^\star} = 1$	$enable_{der_{filter}}^{(z^\star,\hat{z})\rightarrow\dot{x}^\star} = False$ $N_d^{(z^\star,\hat{x})\rightarrow\dot{z}^\star} = N.R$ $enable_{ref_{filter}}^{(z^\star,\hat{z})\rightarrow\dot{x}^\star} = False$ $N_r^{(z^\star,\hat{z})\rightarrow\dot{z}^\star} = N.R.$	$enable_{anti_{windup}}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = False$ $K_{aw}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = N.R.$

Table 5-3: Horizontal speed controller module default parameters (Simulation Pixhawk autopilot)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{hsc} = 0.033s$	$K_p^{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v} = 0.253$ $K_d^{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v} = -0.025$ $K_i^{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v} = 0.115$ $b^{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v} = 0.55$ $c^{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v} = 1.73$ $K_p^{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v} = 0.253$ $K_d^{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v} = -0.025$ $K_i^{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v} = 0.115$ $b^{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v} = 0.55$ $c^{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v} = 1.73$	$U_{Min}^{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v} = -0.2$ $U_{Max}^{(\dot{x}^\star,\hat{\dot{x}})\rightarrow\theta_v} = 0.2$ $U_{Min}^{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v} = -0.2$ $U_{Max}^{(\dot{y}^\star,\hat{\dot{y}})\rightarrow\phi_v} = 0.2$	$enable_{der_{filter}}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = True$ $N_d^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 3.439$ $enable_{ref_{filter}}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = False$ $N_r^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = N.R.$ $enable_{der_{filter}}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = True$ $N_d^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 3.439$ $enable_{ref_{filter}}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = False$ $N_r^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = N.R.$	$enable_{anti_{windup}}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = True$ $K_{aw}^{(x^\star,\hat{x})\rightarrow\dot{x}^\star} = 10.$ $enable_{anti_{windup}}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = True$ $K_{aw}^{(y^\star,\hat{y})\rightarrow\dot{y}^\star} = 10.$

Table 5-4: Vertical speed controller module default parameters (Simulation Pixhawk autopilot)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{vpc} = 0.033s$	$K_p^{(\dot{z}^\star,\hat{z})\to T_v} = 2.014$ $K_d^{(\dot{z}^\star,\hat{z})\to T_v} = -0.331$ $K_i^{(\dot{z}^\star,\hat{z})\to T_v} = 0.502$ $b^{(\dot{z}^\star,\hat{z})\to T_v} = 0.574$ $c^{(\dot{z}^\star,\hat{z})\to T_v} = 1.298$	$U_{Min}^{(\dot{z}^\star,\hat{z})\to T_v} = 0$ $U_{Max}^{(\dot{z}^\star,\hat{z})\to T_v} = 1$	$enable_{der_{filter}}^{(\dot{z}^\star,\hat{z})\to T_v} = True$ $N_d^{(\dot{z}^\star,\hat{z})\to T_v} = 2.29$ $enable_{ref_{filter}}^{(\dot{z}^\star,\hat{z})\to T_v} = False$ $N_r^{(\dot{z}^\star,\hat{z})\to T_v} = N.R.$	$enable_{anti_{windup}}^{(\dot{z}^\star,\hat{z})\to T_v} = True$ $K_{aw}^{(\dot{z}^\star,\hat{z})\to T_v} = 10$

Table 5-5: Yaw controller module default parameters (Simulation Pixhawk autopilot)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{\psi c} = 0.1s$	$K_p^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = 0.15$ $K_d^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = 0$ $K_i^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = 0$ $b^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = 1$ $c^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = 0$	$U_{Min}^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = -0.4$ $U_{Max}^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = 0.4$	$enable_{der_{filter}}^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = False$ $N_d^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = N.R.$ $enable_{ref_{filter}}^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = False$ $N_r^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = N.R.$	$enable_{anti_{windup}}^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = False$ $K_{aw}^{(\psi^\star,\hat{\psi})\to\dot{\psi}^\star} = N.R.$

Table 5-6: Perception module default parameters (ArUco marker has been used)

Ideal centers and desired distance	Camera parameters: Front image (48x640) Bottom image (480x640)	Marker parameters	Rotation matrix from camera to robot frame
${}^f u_5^\star = 240$ ${}^f v_5^\star = 320$ ${}^f d_{mm}^\star = 1000$ ${}^b u_5^\star = 240$ ${}^b v_5^\star = 320$ ${}^b d_{mm}^\star = 750$	${}^f f_{mm} = 7.6$ ${}^f s_{h_{mm}} = 9.9382$ ${}^f im_{h_{pix}} = 480$ ${}^b f_{mm} = 7.6$ ${}^b s_{h_{mm}} = 7.938$ ${}^b im_{h_{pix}} = 480$	${}^f side_{mm} = 170$ ${}^f ID^\star = 12$ ${}^b side_{mm} = 170$ ${}^b ID^\star = 9$	$R_{C_f}^R = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$ $R_{C_b}^R = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$

Table 5-7: Image state estimator module default parameters (ArUco marker has been used)

Estimation approach	Kalman filter approach default parameters
$selector = 1$ (0 = Static approach, 1 = Kalman filter approach)	$T_s = 0.033s$ $Q_{T_{4\times4}} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}$ $R_{T_{2\times2}} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$

Table 5-8: Image based visual servo module default parameters

Safety and saturation constraint variables	IBVS default parameters
$safety\_counter\_threshold = 80$ $Min = -1$ $Max = 1$	${}^{f}interaction_{matrix} = 0$ ${}^{f}\lambda_{3\times3} = \begin{bmatrix} 0.6 & 0 & 0 \\ 0 & 0.6 & 0 \\ 0 & 0 & 0.4 \end{bmatrix}$ ${}^{b}interaction_{matrix} = 0$ ${}^{b}\lambda_{3\times3} = \begin{bmatrix} 0.7 & 0 & 0 \\ 0 & 0.7 & 0 \\ 0 & 0 & 0.15 \end{bmatrix}$

5-1-2 Simulation experiment results

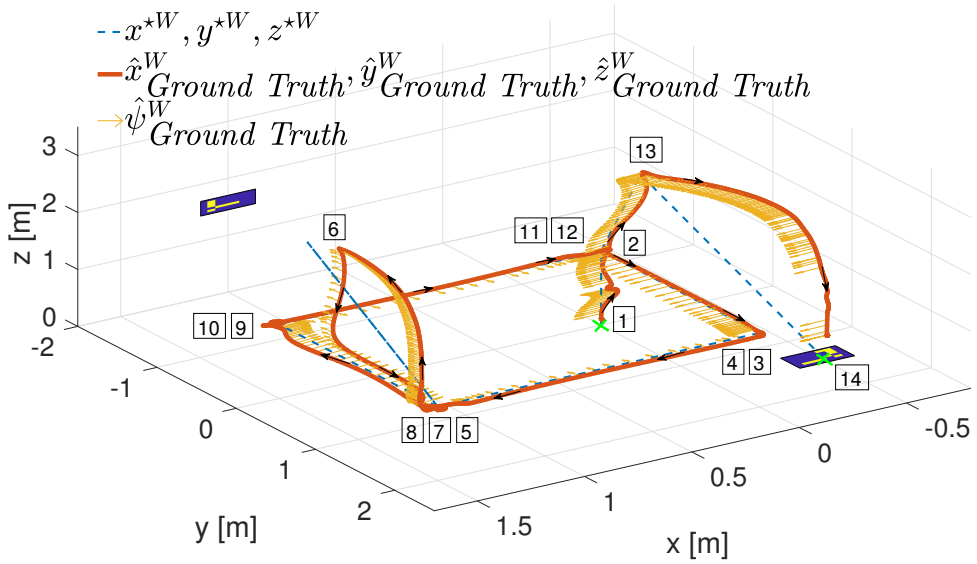


Figure 5-1: The figure shows the quadrotor's $(3D)$ trajectory associated to the quadrotor's mission 4-1. The simulation is performed in the Gazebo simulation environment using the Pixhawk Software-In-The-Loop. The ground truth data are used to estimate the quadrotor's states (position,velocity,acceleration,orientation) expressed in world coordinate frame. The numbers appearing on the figure are used to indicate the mission task's number that the quadrotor is facing.

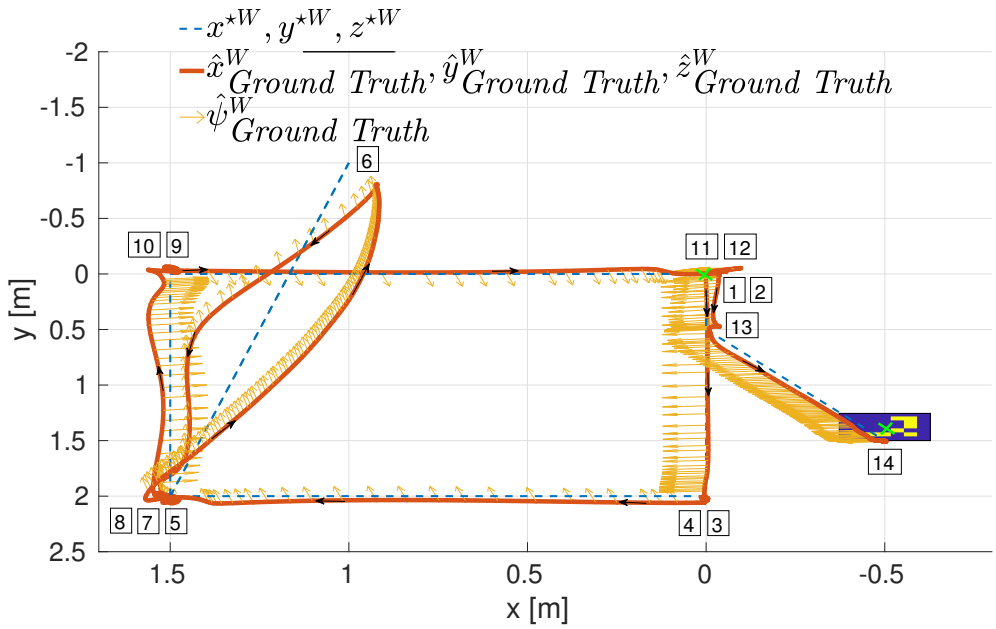


Figure 5-2: The figure shows the $(2D)$ quadrotor's trajectory associated to the quadrotor mission 4-1 which it stands for how the quadrotor moves along the (x) and (y) direction of the world coordinate frame. The numbers appearing on the figure are used to indicate the mission task's number that the quadrotor is facing.

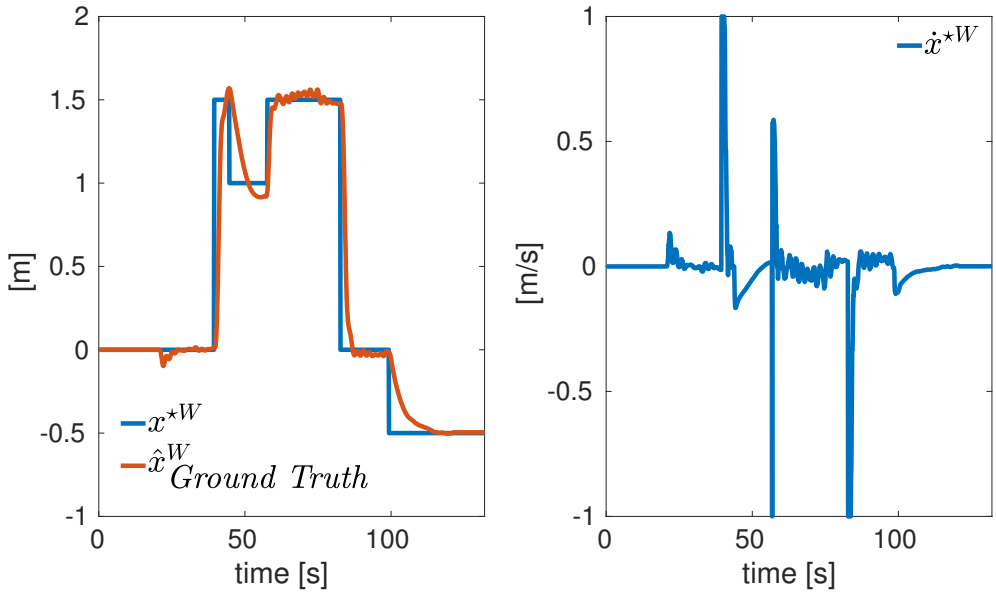


Figure 5-3: The figure shows on the left side the quadrotor's ground truth estimated $(\hat{x}^W)$ position (red line) with respect to the desired reference $(x^{\star W})$ (blue line). On the right side it is shown the output of the horizontal position controller module $(\dot{x}^{\star W})$ representing the desired reference velocity along the (x) direction of the quadrotor expressed in world coordinate frame.

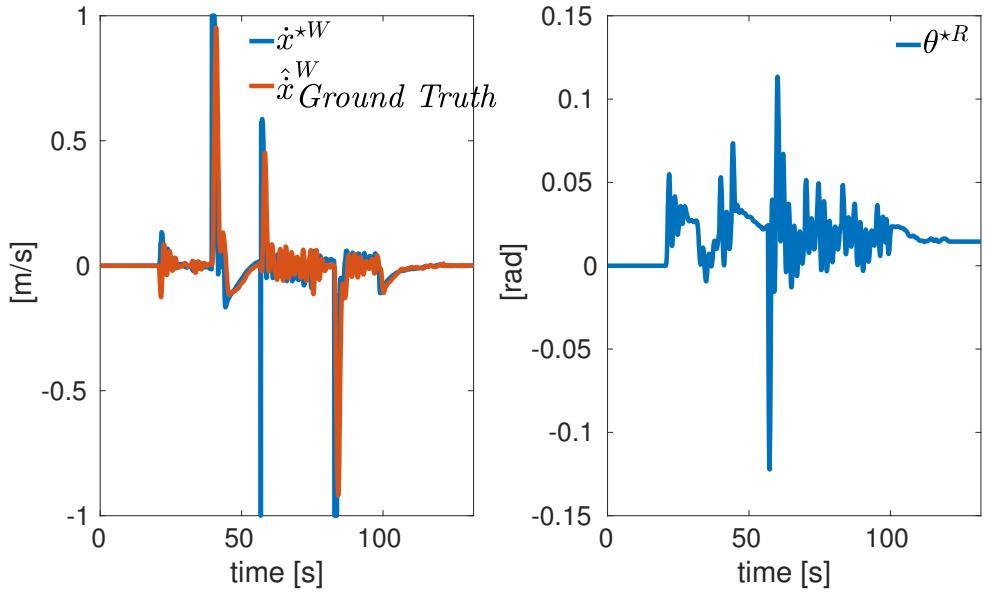


Figure 5-4: The figure shows on the left side the quadrotor's ground truth estimated $(\hat{\dot{x}}^{W})$ velocity (red line) with respect to the desired reference $(\dot{x}^{\star W})$ (blue line). On the right side it is shown the output of the horizontal speed controller module $(\theta^{\star R})$ representing the desired reference pitch angle of the quadrotor expressed in robot coordinate frame.

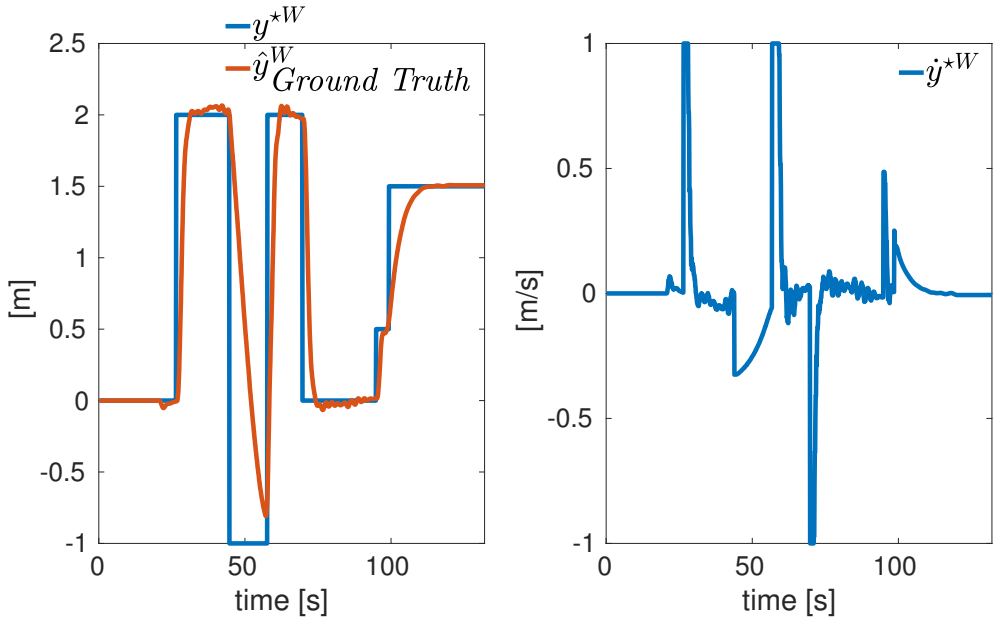


Figure 5-5: The figure shows on the left side the quadrotor's ground truth estimated $(\hat{y}^{W})$ position (red line) with respect to the desired reference $(y^{\star W})$ (blue line). On the right side it is shown the output of the horizontal position controller module $(\dot{y}^{\star W})$ representing the desired reference velocity along the (y) direction of the quadrotor expressed in world coordinate frame.

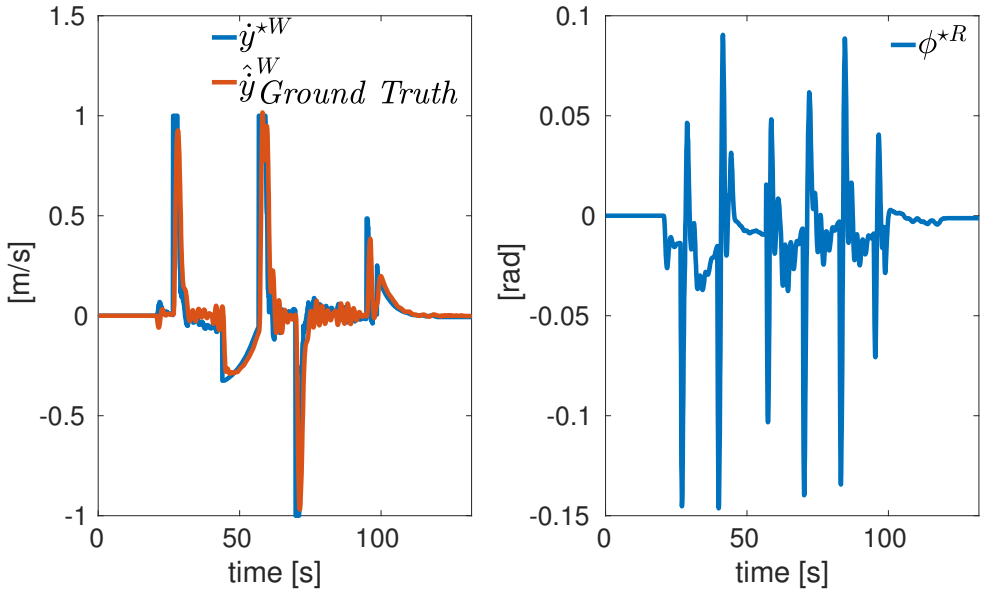


Figure 5-6: The figure shows on the left side the quadrotor's ground truth estimated $(\hat{\dot{y}}^{W})$ velocity (red line) with respect to the desired reference $(\dot{y}^{\star W})$ (blue line). On the right side it is shown the output of the horizontal speed controller module $(\phi^{\star R})$ representing the desired reference roll angle of the quadrotor expressed in robot coordinate frame.

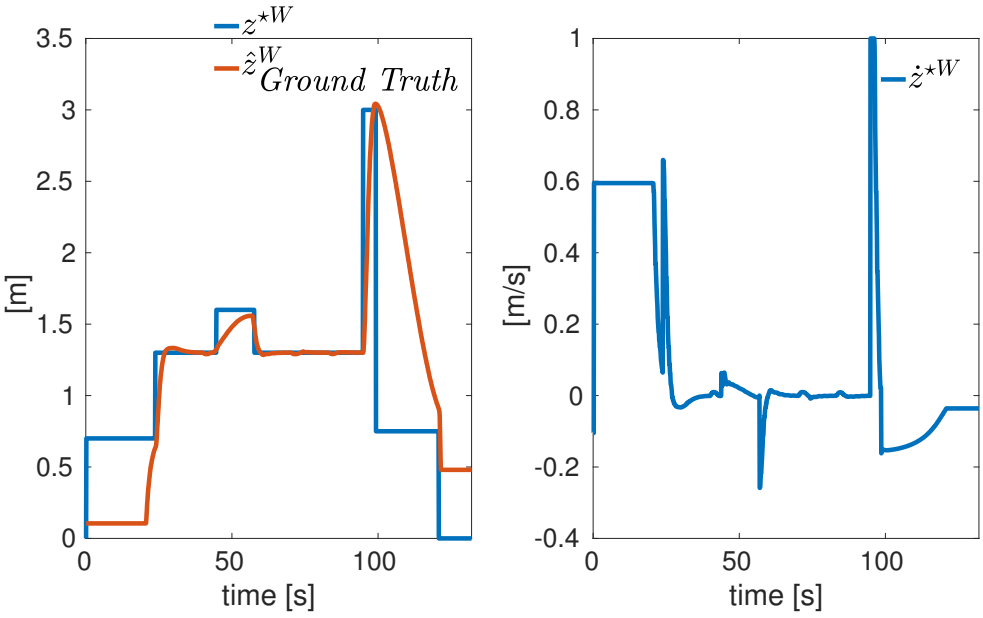


Figure 5-7: The figure shows on the left side the quadrotor's ground truth estimated $(\hat{z}^{W})$ position (red line) with respect to the desired reference $(z^{\star W})$ (blue line). On the right side it is shown the output of the vertical position controller module $(\dot{z}^{\star W})$ representing the desired reference velocity along the (z) direction of the quadrotor expressed in world coordinate frame.

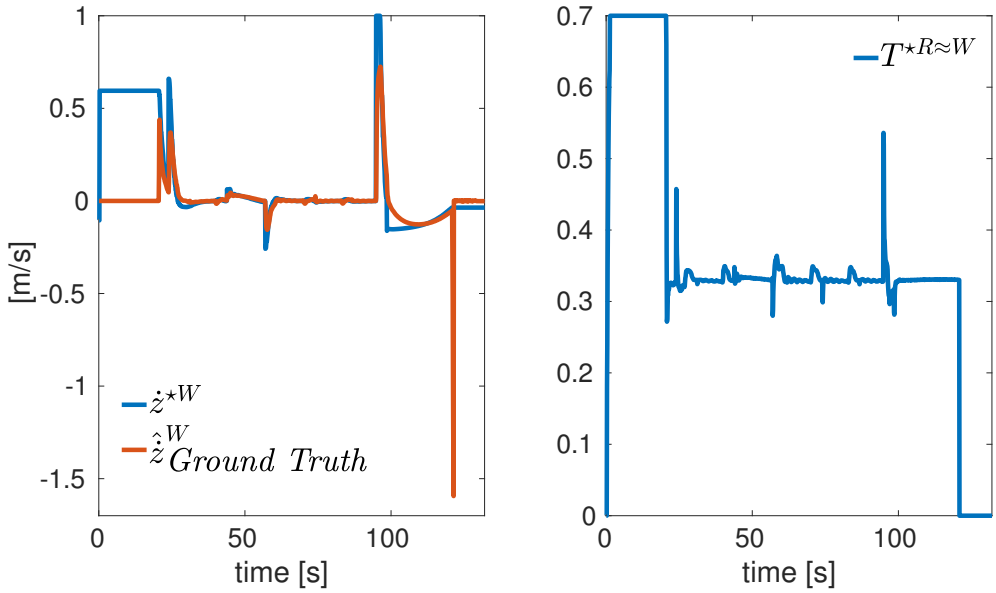


Figure 5-8: The figure shows on the left side the quadrotor's ground truth estimated $(\hat{\dot{z}}^{W})$ velocity (red line) with respect to the desired reference $(\dot{z}^{\star W})$ (blue line). On the right side it is shown the output of the vertical speed controller module $(T^{\star R \approx W})$ representing the desired reference thrust along the (z) direction of the quadrotor expressed in robot coordinate frame.

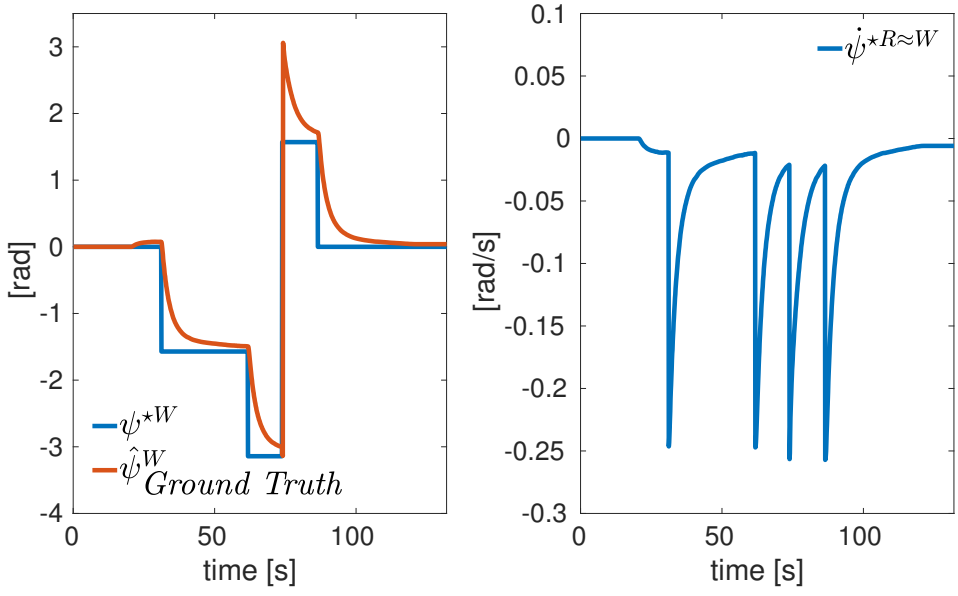


Figure 5-9: The figure shows on the left side the quadrotor's ground truth estimated $(\hat{\psi}^{W})$ angle (red line) with respect to the desired reference $(\psi^{\star W})$ (blue line). On the right side it is shown the output of the yaw controller module $(\dot{\psi}^{\star W})$ representing the desired reference yaw angle of the quadrotor expressed in world coordinate frame.

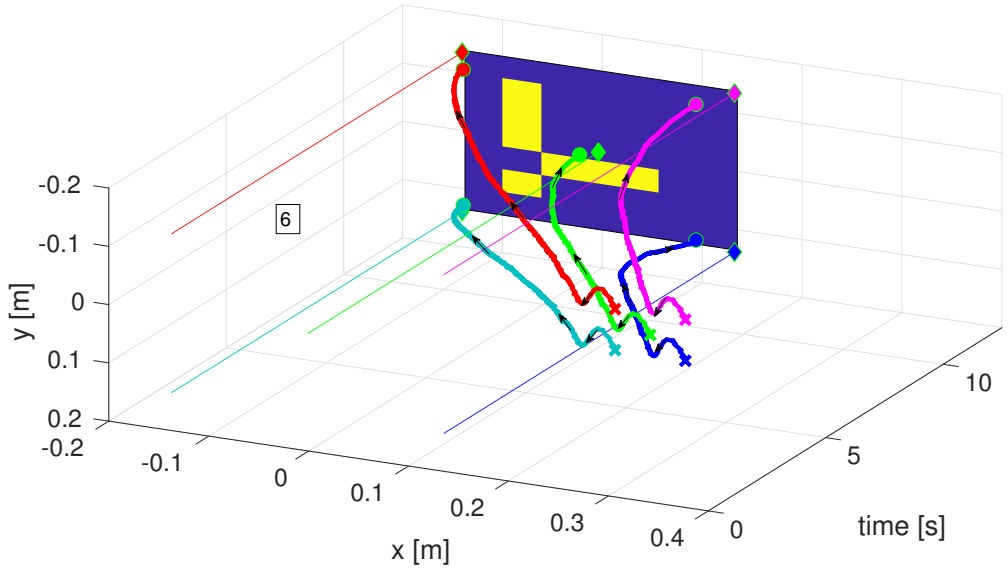


Figure 5-10: The figure shows the evolution in time of the estimated marker's corners and center (start $\times$, end $\circ$) towards the desired ones ($\diamond$). The number 6 appearing in the figure it is used to underline that these results belong to the quadrotor task number 6. The latter consists in approaching a static visual marker using the front camera up to a desired distance which it is equivalent to drive on the current $(2D)$ image plane the estimated corners towards the desired ones.

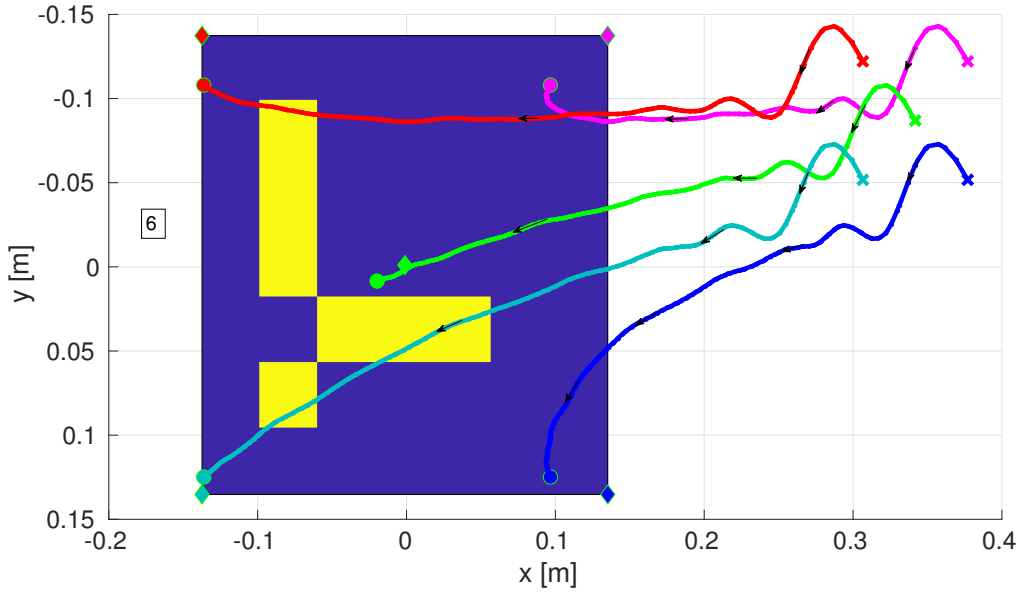


Figure 5-11: The figure shows the trajectories of the estimated corners and center (start $\times$, end $\circ$) towards the desired ones ($\diamond$) on the $(2D)$ image coordinate plane. This result is associated to quadrotor task number 6.

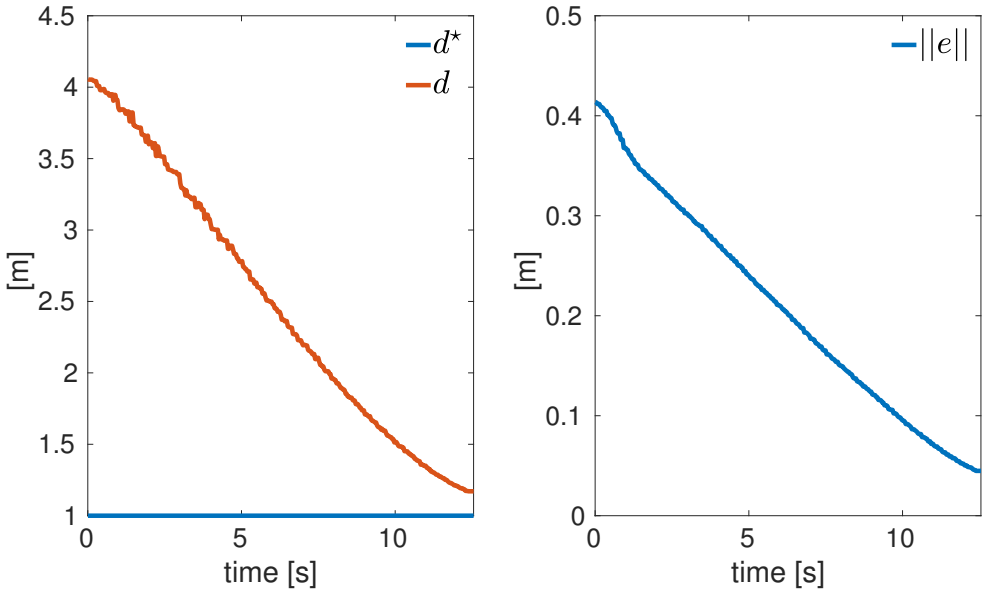


Figure 5-12: The figure shows on the left side the estimated distance (d red line) between quadrotor and ArUco marker computed using Eq. (3-9) with respect to the desired chosen value ($d^\star$ blue line). On the right side it is shown the evolution in time of the norm of the IBVS error given in Eq. (3-25). The result shows in this figure belongs to the mission task number 6 in which the quadrotor uses the front camera to approach a static object up to a desired distance (equal to 1m), minimizing the error ($|e|$) between the estimated and the desired marker's corners and center.

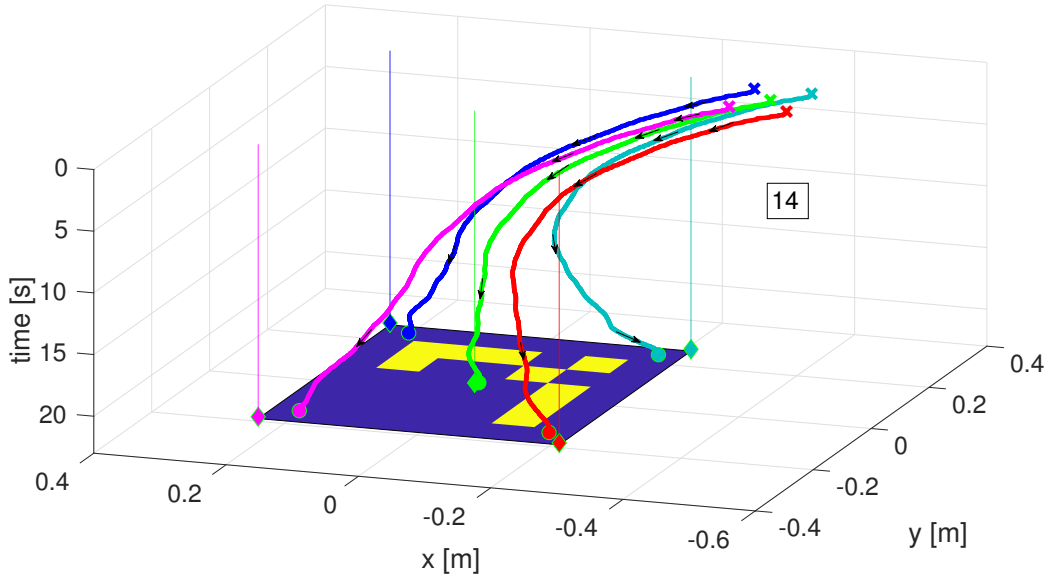


Figure 5-13: The figure shows the evolution in time of the estimated marker's corners and center (start $\times$, end $\circ$) towards the desired ones ($\diamond$). This figure is associated to the mission task number 14. The latter consists in making the quadrotor approaching a static visual marker using the bottom camera up to a desired distance. This task is equivalent to say that the quadrotor is autonomously landing on a static platform on which the ArUco visual marker is placed. Indeed when the quadrotor reach the desired a land command is sent.

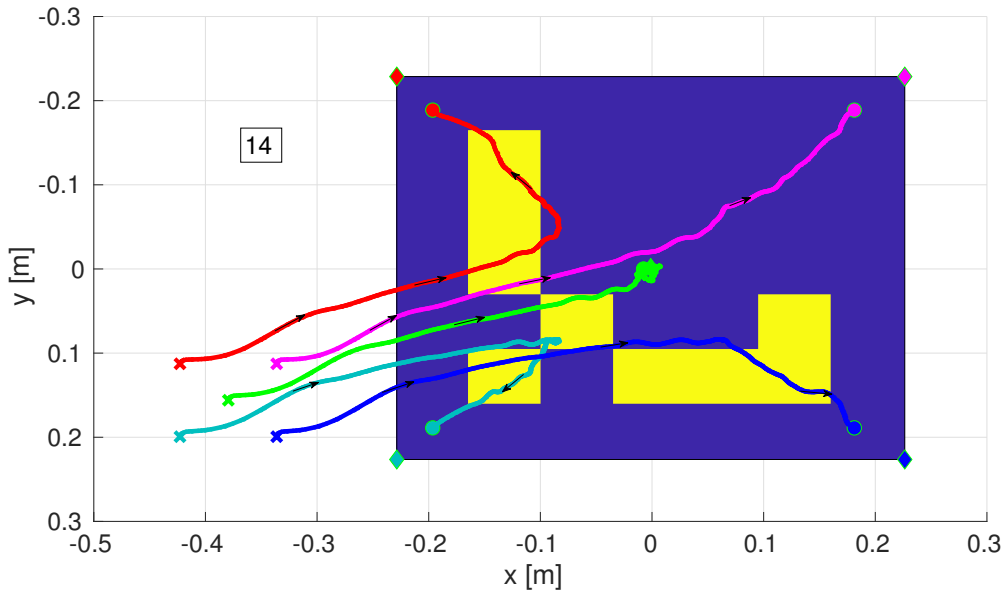


Figure 5-14: The figure shows the trajectories of the estimated corners and center (start $\times$, end $\circ$) towards the desired ones ($\diamond$) on the $(2D)$ image coordinate plane. In particular it represents how the estimated marker's corners and center move towards the desired ones. This figure is associated to the mission task number 14.

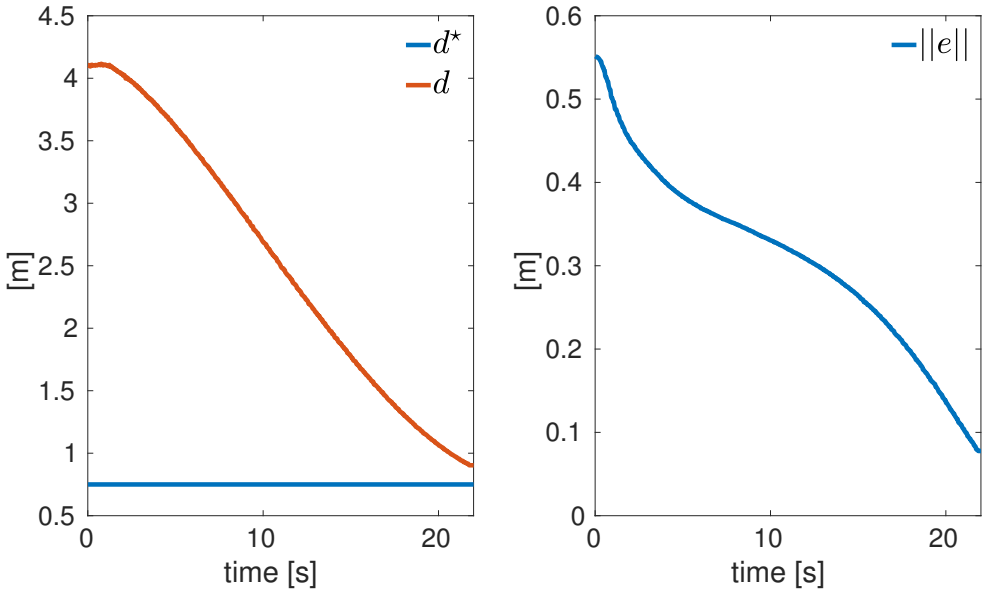


Figure 5-15: The figure shows on the left side the estimated distance (d red line) between quadrotor and ArUco marker computed using Eq. (3-9) with respect to the desired chosen value ($d^\star$ blue line). On the right side it is shown the evolution in time of the norm of the IBVS error given in Eq. (3-25). The result shows in this figure belongs to the mission task number 14 in which the quadrotor uses the bottom camera to approach a static object up to a desired distance (equal to 0.75m), minimizing the error ($|e|$) between the estimated and the desired marker's corners and center. When this distance is reached the quadrotor lands on the platform.

5-2 Real flight experiment

The video associated to the real flight experiment presented in this section is available here.

https://www.youtube.com/watch?v=T6Ep4t_QX6M

Another video showing how the quadrotor is able to follow using the front camera a visual marker is provided here (https://www.youtube.com/watch?v=GssfIm8woMg) and here (https://www.youtube.com/watch?v=Eot1V2JZUf4).

5-2-1 Default parameters used in the real experiment (AR Drone 2.0))

The navigation controller framework and the vision based planner parameters that have been used to perform the fully autonomous mission described in 4-1 are provided. The description of their meaning together with the explanation of how they are derived is given in chapter 2 and 3 . Available in Appendix B-2 are also the results associated to the experiments that have been done to properly select the controller parameters of the navigation controller framework.

Table 5-9: Horizontal position controller module default parameters (Parrot AR. DRONE 2.0)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{hpc} = 0.1s$	$K_p^{(x^\star,\hat{x})\to\dot{x}^\star} = 0.5$ $K_d^{(x^\star,\hat{x})\to\dot{x}^\star} = 0$ $K_i^{(x^\star,\hat{x})\to\dot{x}^\star} = 0$ $b^{(x^\star,\hat{x})\to\dot{x}^\star} = 1$ $c^{(x^\star,\hat{x})\to\dot{x}^\star} = 0$ $K_p^{(y^\star,\hat{y})\to\dot{y}^\star} = 0.5$ $K_d^{(y^\star,\hat{y})\to\dot{y}^\star} = 0$ $K_i^{(y^\star,\hat{y})\to\dot{y}^\star} = 0$ $b^{(y^\star,\hat{y})\to\dot{y}^\star} = 1$ $c^{(y^\star,\hat{y})\to\dot{y}^\star} = 0$	$U_{Min}^{(x^\star,\hat{x})\to\dot{x}^\star} = -1$ $U_{Max}^{(x^\star,\hat{x})\to\dot{x}^\star} = 1$ $U_{Min}^{(y^\star,\hat{y})\to\dot{y}^\star} = -1$ $U_{Max}^{(y^\star,\hat{y})\to\dot{y}^\star} = 1$	$enable_{der_{filter}}^{(x^\star,\hat{x})\to\dot{x}^\star} = True$ $N_d^{(x^\star,\hat{x})\to\dot{x}^\star} = 17.3$ $enable_{ref_{filter}}^{(x^\star,\hat{x})\to\dot{x}^\star} = False$ $N_r^{(x^\star,\hat{x})\to\dot{x}^\star} = N.R.$ $enable_{der_{filter}}^{(y^\star,\hat{y})\to\dot{y}^\star} = True$ $N_d^{(y^\star,\hat{y})\to\dot{y}^\star} = 17.3$ $enable_{ref_{filter}}^{(y^\star,\hat{y})\to\dot{y}^\star} = False$ $N_r^{(y^\star,\hat{y})\to\dot{y}^\star} = N.R.$	$enable_{anti_{windup}}^{(x^\star,\hat{x})\to\dot{x}^\star} = False$ $K_{aw}^{(x^\star,\hat{x})\to\dot{x}^\star} = N.R.$ $enable_{anti_{windup}}^{(y^\star,\hat{y})\to\dot{y}^\star} = False$ $K_{aw}^{(y^\star,\hat{y})\to\dot{y}^\star} = N.R.$

Table 5-10: Vertical position controller module default parameters (Parrot AR. DRONE 2.0)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{vpc} = 0.1s$	$K_p^{(z^\star,\hat{z})\to\dot{z}^\star} = 1$ $K_d^{(z^\star,\hat{z})\to\dot{z}^\star} = 0$ $K_i^{(z^\star,\hat{z})\to\dot{z}^\star} = 0$ $b^{(z^\star,\hat{z})\to\dot{z}^\star} = 1$ $c^{(z^\star,\hat{z})\to\dot{z}^\star} = 0$	$U_{Min}^{(z^\star,\hat{z})\to\dot{z}^\star} = -1$ $U_{Max}^{(z^\star,\hat{z})\to\dot{z}^\star} = 1$	$enable_{der_{filter}}^{(z^\star,\hat{z})\to\dot{x}^\star} = False$ $N_d^{(z^\star,\hat{x})\to\dot{z}^\star} = N.R$ $enable_{ref_{filter}}^{(z^\star,\hat{z})\to\dot{x}^\star} = False$ $N_r^{(z^\star,\hat{z})\to\dot{z}^\star} = N.R.$	$enable_{anti_{windup}}^{(x^\star,\hat{x})\to\dot{x}^\star} = False$ $K_{aw}^{(x^\star,\hat{x})\to\dot{x}^\star} = N.R.$

Table 5-11: Horizontal speed controller module default parameters (Parrot AR. DRONE 2.0)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{hsc} = 0.033s$	$K_p^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} = 0.319$ $K_d^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} = -0.026$ $K_i^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} = 0.181$ $b^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} = 0.549$ $c^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} = 1.729$ $K_p^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} = 0.367$ $K_d^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} = -0.026$ $K_i^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} = 0.238$ $b^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} = 0.548$ $c^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} = 1.723$	$U_{Min}^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} = -0.2$ $U_{Max}^{(\dot{x}^\star,\hat{\dot{x}})\to\theta_v} = 0.2$ $U_{Min}^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} = -0.2$ $U_{Max}^{(\dot{y}^\star,\hat{\dot{y}})\to\phi_v} = 0.2$	$enable_{der_{filter}}^{(x^\star,\hat{x})\to\dot{x}^\star} = True$ $N_d^{(x^\star,\hat{x})\to\dot{x}^\star} = 4.337$ $enable_{ref_{filter}}^{(x^\star,\hat{x})\to\dot{x}^\star} = False$ $N_r^{(x^\star,\hat{x})\to\dot{x}^\star} = N.R.$ $enable_{der_{filter}}^{(y^\star,\hat{y})\to\dot{y}^\star} = True$ $N_d^{(y^\star,\hat{y})\to\dot{y}^\star} = 4.979$ $enable_{ref_{filter}}^{(y^\star,\hat{y})\to\dot{y}^\star} = False$ $N_r^{(y^\star,\hat{y})\to\dot{y}^\star} = N.R.$	$enable_{anti_{windup}}^{(x^\star,\hat{x})\to\dot{x}^\star} = True$ $K_{aw}^{(x^\star,\hat{x})\to\dot{x}^\star} = 10.$ $enable_{anti_{windup}}^{(y^\star,\hat{y})\to\dot{y}^\star} = True$ $K_{aw}^{(y^\star,\hat{y})\to\dot{y}^\star} = 10.$

Table 5-12: Yaw controller module default parameters (Parrot AR. DRONE 2.0)

Sampling time	2DOF PID gains	Saturations boundaries	Derivative error and reference filter gains	Anti windup
$T_s^{\psi c} = 0.1s$	$K_p^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = 1.2$ $K_d^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = 0$ $K_i^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = 0$ $b^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = 1$ $c^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = 0$	$U_{Min}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = -0.4$ $U_{Max}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = 0.4$	$enable_{der_{filter}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = False$ $N_d^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = N.R.$ $enable_{ref_{filter}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = False$ $N_r^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = N.R.$	$enable_{anti_{windup}}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = False$ $K_{aw}^{(\psi^\star,\hat{\psi})\rightarrow\dot{\psi}^\star} = N.R.$

Table 5-13: Perception module default parameters (ArUco marker has been used)

Ideal centers and desired distance	Camera parameters: Front image (360x640) Bottom image (360x640)	Marker parameters	Rotation matrix from camera to robot frame
${}^f u_5^\star = 180$ ${}^f v_5^\star = 320$ ${}^f d_{mm}^\star = 1000$ ${}^b u_5^\star = 180$ ${}^b v_5^\star = 320$ ${}^b d_{mm}^\star = 750$	${}^f f_{mm} = 7.6$ ${}^f s_{h_{mm}} = 9.9382$ ${}^f im_{h_{pix}} = 360$ ${}^b f_{mm} = 7.6$ ${}^b s_{h_{mm}} = 7.938$ ${}^b im_{h_{pix}} = 360$	${}^f side_{mm} = 255$ ${}^f ID^\star = 13$ ${}^b side_{mm} = 255$ ${}^b ID^\star = 2$	$R_{C_f}^R = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$ $R_{C_b}^R = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$

Table 5-14: Image state estimator module default parameters (ArUco marker has been used)

Estimation approach	Kalman filter approach default parameters
$selector = 1$ (0 = Static approach, 1 = Kalman filter approach)	$T_s = 0.033s$ $Q_{T_{4\times4}} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}$ $R_{T_{2\times2}} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$

Table 5-15: Image based visual servo module default parameters

Safety and saturation constraint variables	IBVS default parameters
$safety\_counter\_threshold = 40$ $Min = -1$ $Max = 1$	${}^{f}interaction_{matrix} = 2$ ${}^{f}\lambda_{3\times3} = \begin{bmatrix} 0.65 & 0 & 0 \\ 0 & 0.45 & 0 \\ 0 & 0 & 0.6 \end{bmatrix}$ ${}^{b}interaction_{matrix} = 2$ ${}^{b}\lambda_{3\times3} = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.3 \end{bmatrix}$

5-2-2 Real flight experiment results

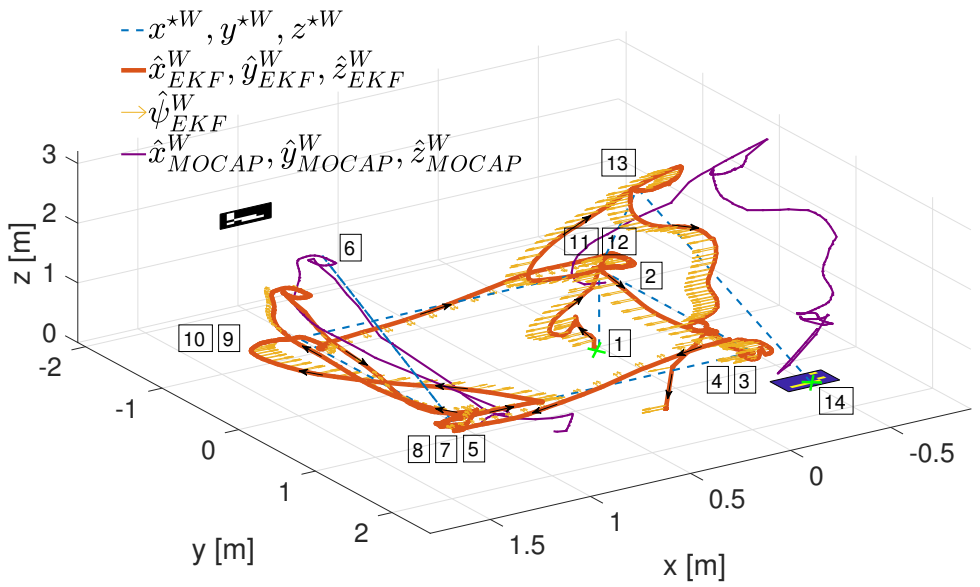


Figure 5-16: The figure shows the quadrotor's $(3D)$ trajectory associated to the quadrotor's real flight mission. The mission is performed using the AR Drone 2.0. An EKF is used to estimate the quadrotor's states (position (red line) ,velocity,acceleration,orientation) expressed in world coordinate frame. The yellow arrows are used to represent the quadrotor estimated yaw angle during the mission. The purple line represents the ground truth data measured during the *Approach_to_an_object* (visual servoing) tasks (6 and 14). The latter shows that the quadrotor is moving correctly toward the goal identified by the dashed blue line although the estimation of the position provided by the EKF (red line) is wrong. This occurs because during task 6 and 14 only EKF velocity measurements and visual information are used. The numbers appearing on the figure are used to indicate the mission task's number that the quadrotor is facing.

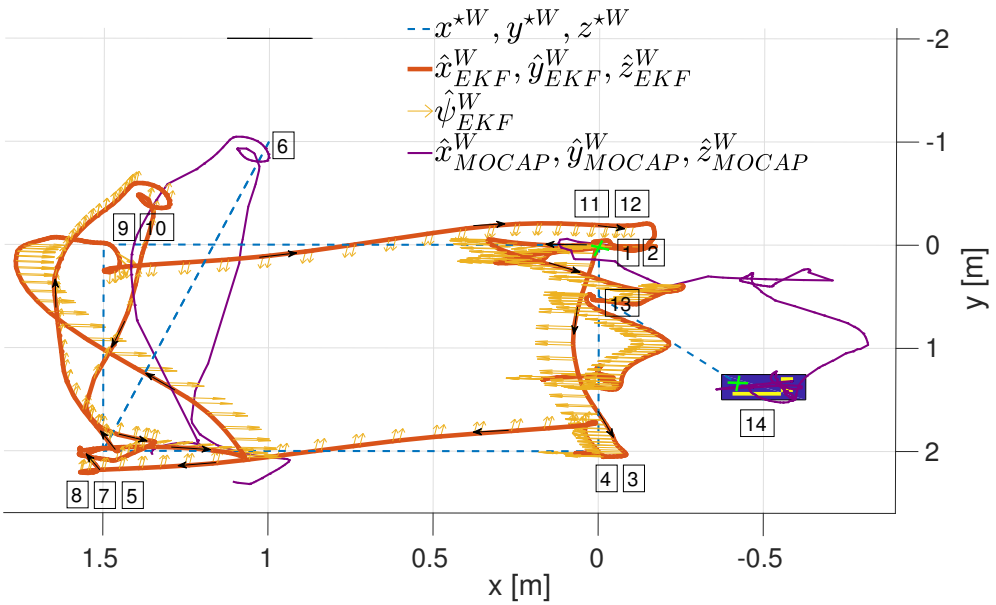


Figure 5-17: The figure shows the $(2D)$ quadrotor's trajectory associated to the quadrotor mission which it stands for how the quadrotor moves along the (x) and (y) direction of the world coordinate frame. The numbers appearing on the figure are used to indicate the mission task's number that the quadrotor is facing.

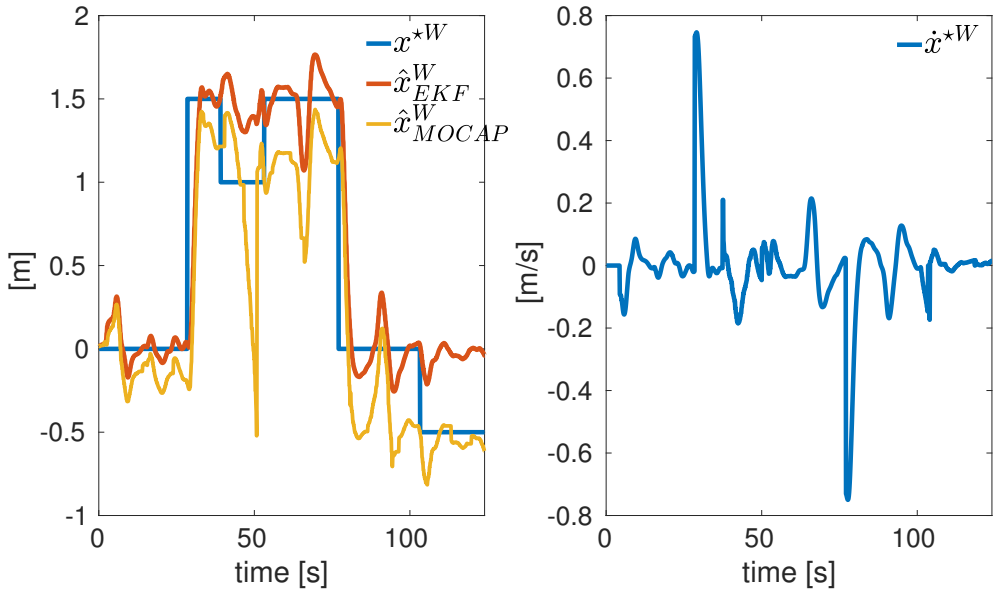


Figure 5-18: The figure shows on the left side the quadrotor's EKF estimated $(\hat{x}^W)$ position (red line) with respect to the desired reference $(x^{\star W})$ (blue line). The ground truth position $(\hat{x}^W_{MOCAP})$ (yellow line) is provided for a comparison purpose. On the right side it is shown the output of the horizontal position controller module $(\dot{x}^{\star W})$ representing the desired reference velocity along the (x) direction of the quadrotor expressed in world coordinate frame.

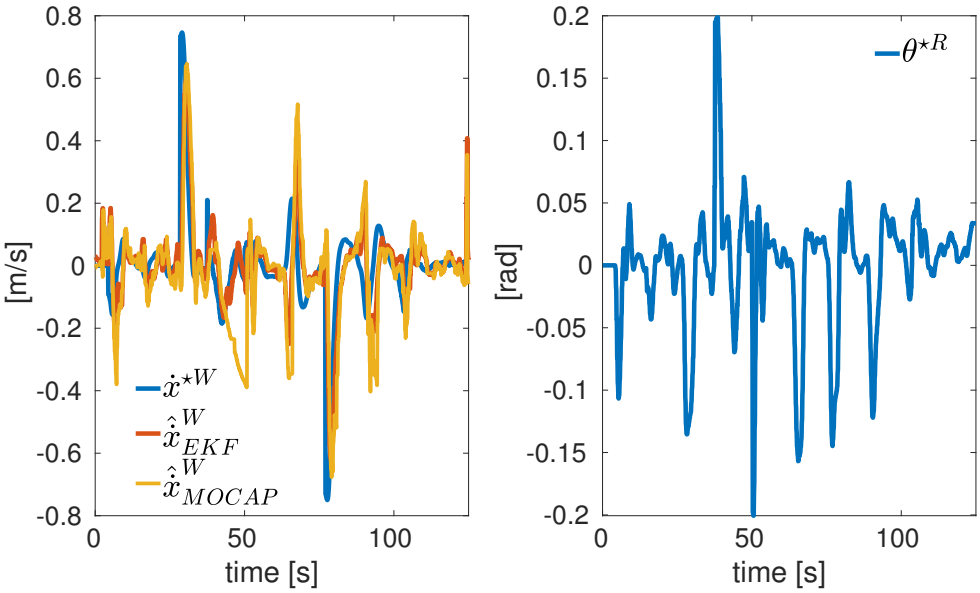


Figure 5-19: The figure shows on the left side the quadrotor's EKF estimated $(\hat{\dot{x}}^{W})$ velocity (red line) with respect to the desired reference $(\dot{x}^{\star W})$ (blue line). The ground truth velocity $(\hat{\dot{x}}^{W}_{MOCAP})$ (yellow line) is provided for a comparison purpose. On the right side it is shown the output of the horizontal speed controller module $(\theta^{\star R})$ representing the desired reference pitch angle of the quadrotor expressed in robot coordinate frame.

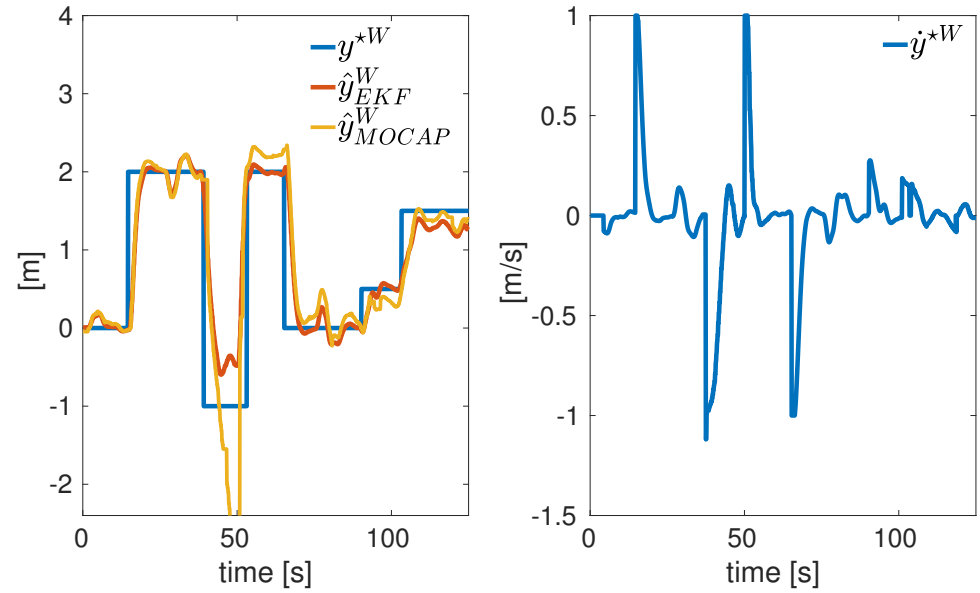


Figure 5-20: The figure shows on the left side the quadrotor's EKF estimated $(\hat{y}^{W})$ position (red line) with respect to the desired reference $(y^{\star W})$ (blue line). The ground truth position $(\hat{y}^{W}_{MOCAP})$ (yellow line) is provided for a comparison purpose. On the right side it is shown the output of the horizontal position controller module $(\dot{y}^{\star W})$ representing the desired reference velocity along the (y) direction of the quadrotor expressed in world coordinate frame.

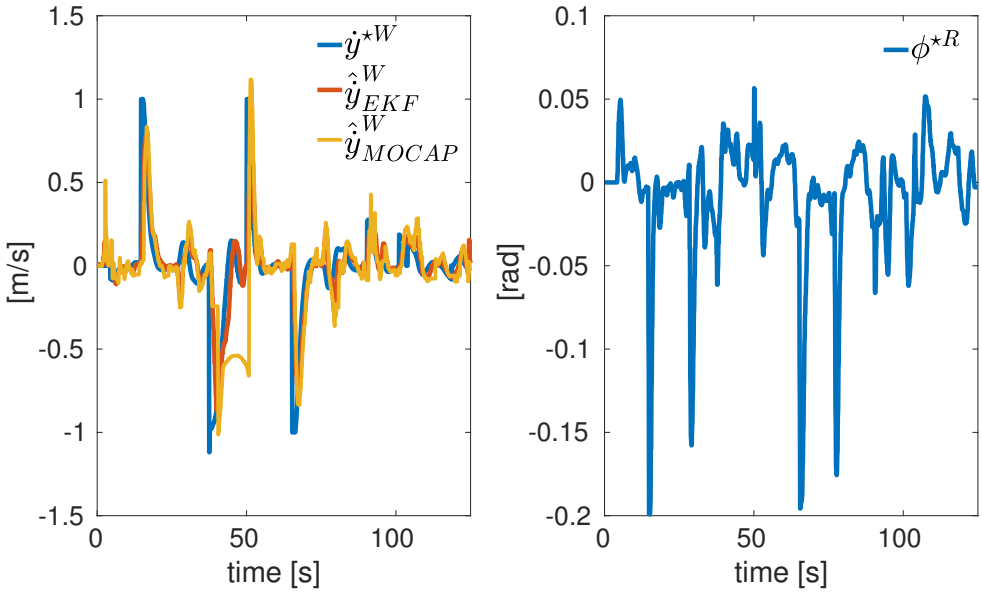


Figure 5-21: The figure shows on the left side the quadrotor's EKF estimated $(\hat{\dot{y}}^{W})$ velocity (red line) with respect to the desired reference $(\dot{y}^{\star W})$ (blue line). The ground truth velocity $(\hat{\dot{y}}^{W}_{MOCAP})$ (yellow line) is provided for a comparison purpose. On the right side it is shown the output of the horizontal speed controller module $(\phi^{\star R})$ representing the desired reference pitch angle of the quadrotor expressed in robot coordinate frame.

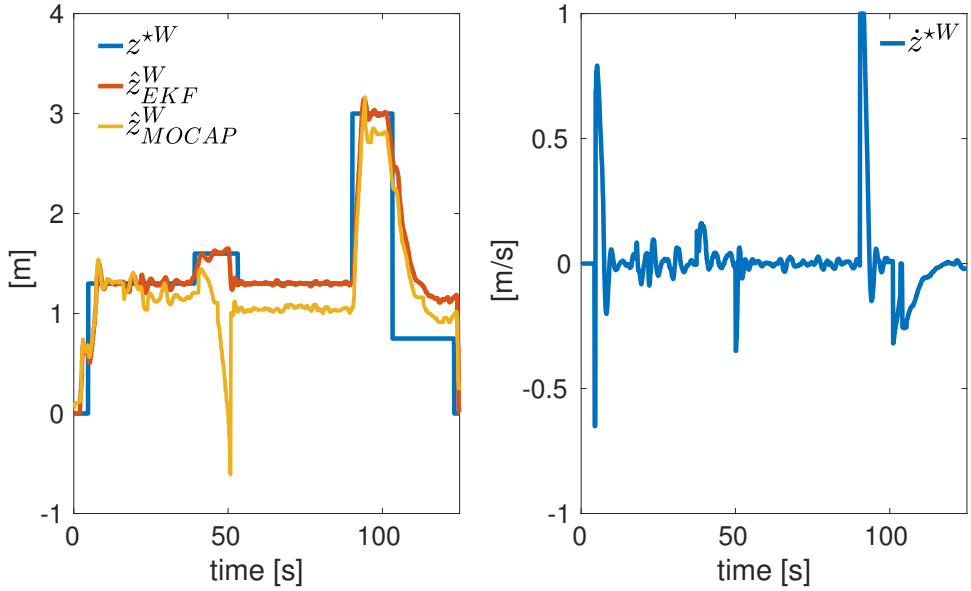


Figure 5-22: The figure shows on the left side the quadrotor's EKF estimated $(\hat{z}^{W})$ position (red line) with respect to the desired reference $(z^{\star W})$ (blue line). The ground truth position $(\hat{z}^{W}_{MOCAP})$ (yellow line) is provided for a comparison purpose. On the right side it is shown the output of the horizontal position controller module $(\dot{z}^{\star W})$ representing the desired reference velocity along the (z) direction of the quadrotor expressed in world coordinate frame.

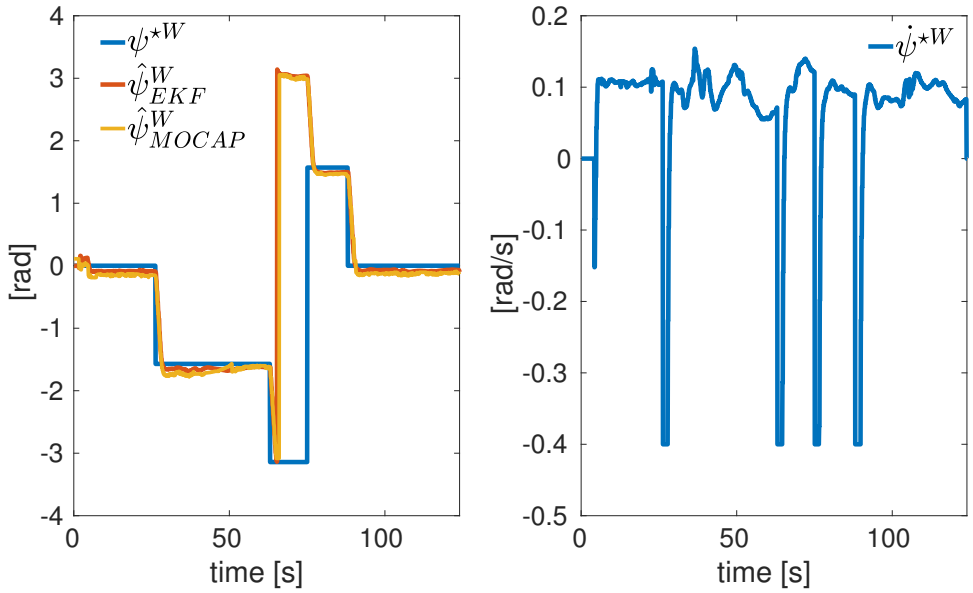


Figure 5-23: The figure shows on the left side the quadrotor's EKF estimated ($\hat{\psi}^{W}$) angle (red line) with respect to the desired reference ($\psi^{\star W}$) (blue line). The ground truth angle ($\hat{\psi}^{W}_{MOCAP}$) (yellow line) is provided for a comparison purpose On the right side it is shown the output of the yaw controller module ($\dot{\psi}^{\star W}$) representing the desired reference yaw angle of the quadrotor expressed in world coordinate frame.

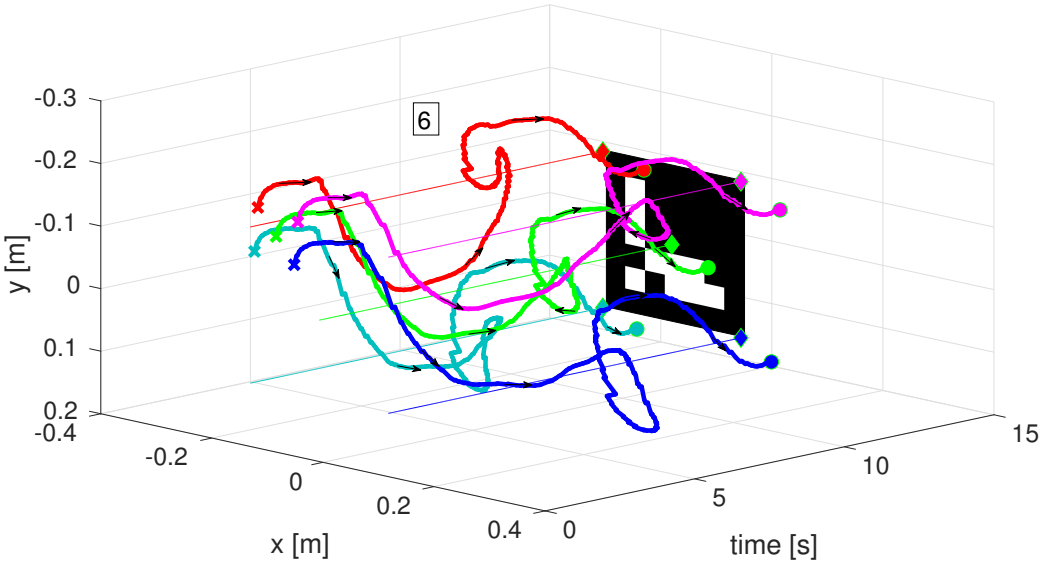


Figure 5-24: The figure shows the evolution in time of the estimated marker's corners and center (start $\times$, end $\circ$) towards the desired ones ($\diamond$). The number 6 appearing in the figure it is used to underline that these results belong to the quadrotor task number 6. The latter consists in approaching a static visual marker using the front camera up to a desired distance which it is equivalent to drive on the current $(2D)$ image plane the estimated corners towards the desired ones. To solve this task front camera image information together with the EKF velocities and yaw angle measurements have been used.

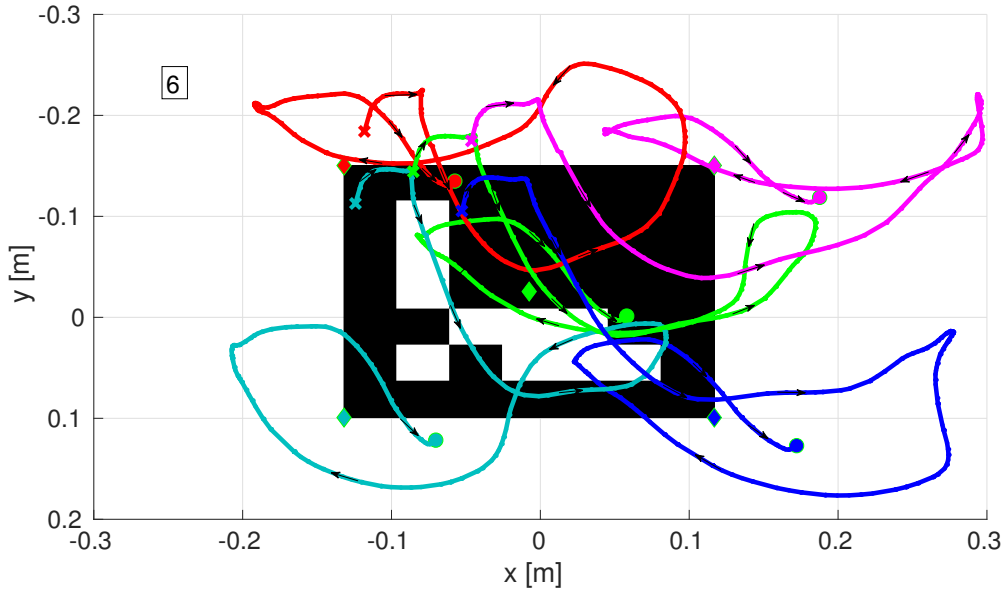


Figure 5-25: The figure shows the trajectories of the estimated corners and center (start $\times$, end $\circ$) towards the desired ones ($\diamond$) on the $(2D)$ image coordinate plane. In particular it represents how the estimated marker corners and center move towards the desired ones.

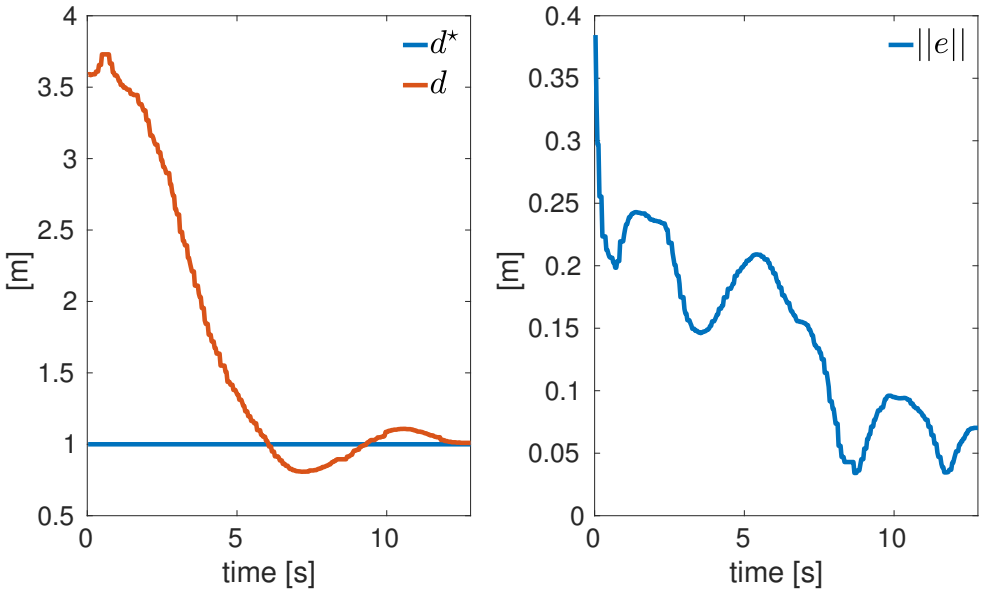


Figure 5-26: The figure shows on the left side the estimated distance (d red line) between quadrotor and ArUco marker computed using Eq. (3-9) with respect to the desired chosen value ($d^\star$ blue line). On the right side it is shown the evolution in time of the norm of the IBVS error given in Eq. (3-25). The result shows in this figure belongs to the mission task number 6 in which the quadrotor uses the front camera to approach a static object up to a desired distance (equal to 1m), minimizing the error ($|e|$) between the estimated and the desired marker's corners and center.

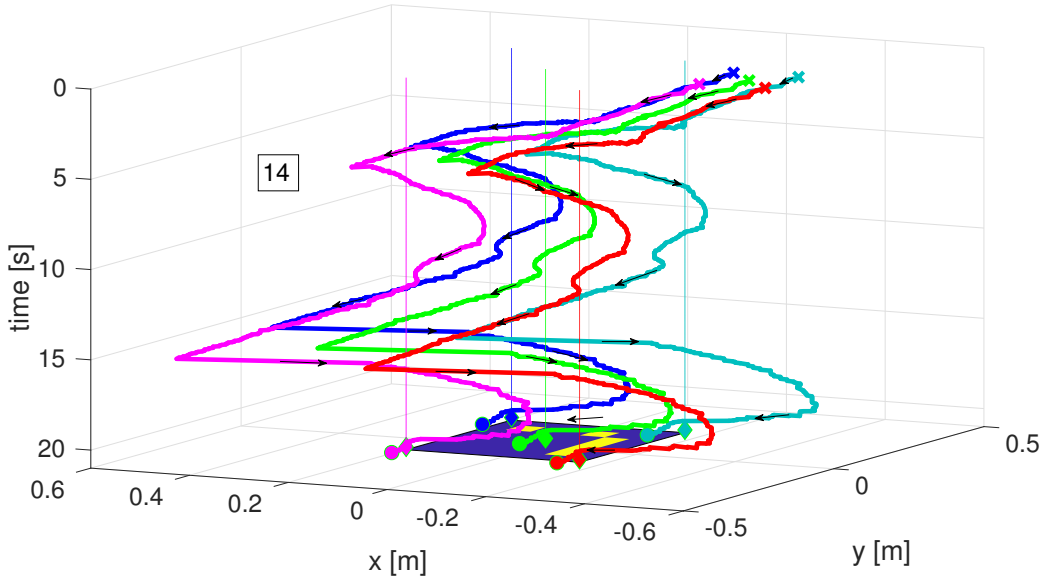


Figure 5-27: The figure shows the evolution in time of the estimated marker's corners and center (start $\times$, end $\circ$) towards the desired ones ($\diamond$) during the quadrotor mission task number 14. The latter consists in making the quadrotor approaching a static visual marker using the bottom camera up to a desired distance.

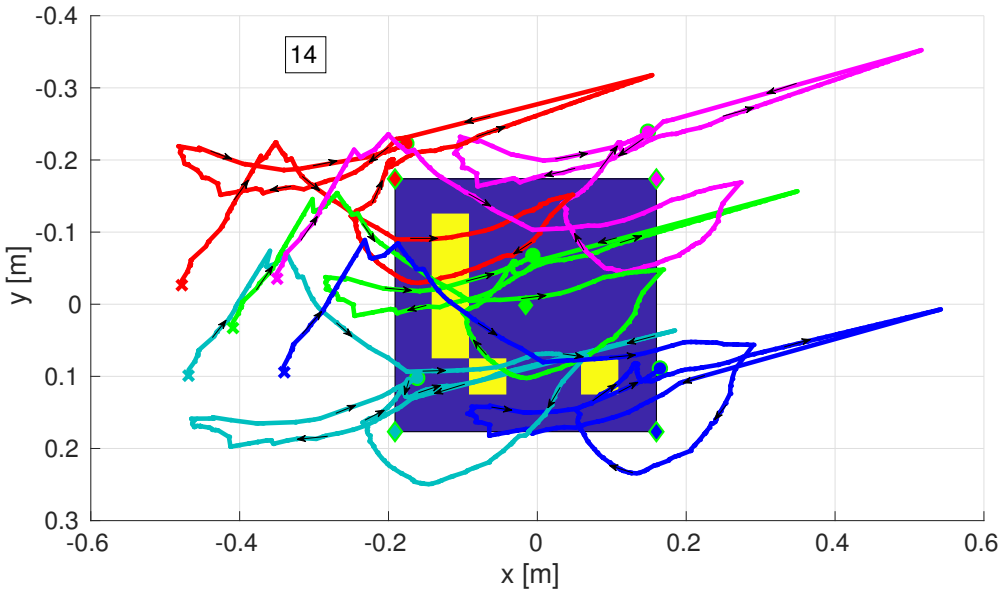


Figure 5-28: The figure shows the trajectories of the estimated points (start $\times$, end $\circ$) towards the desired ones ($\diamond$) on the $(2D)$ image plane during the quadrotor task number 14. The top right figure spikes represent the estimated points predicted by the image state estimator module (Kalman filter with velocity constant model) when a detection is lost. A fast converge to the new detected points is obtained setting the diagonal values of the measurement covariance matrix $(R_{T_{2\times 2}})$ to small values (high confidence in the measurements) (see Table 5-14 pixel unit).

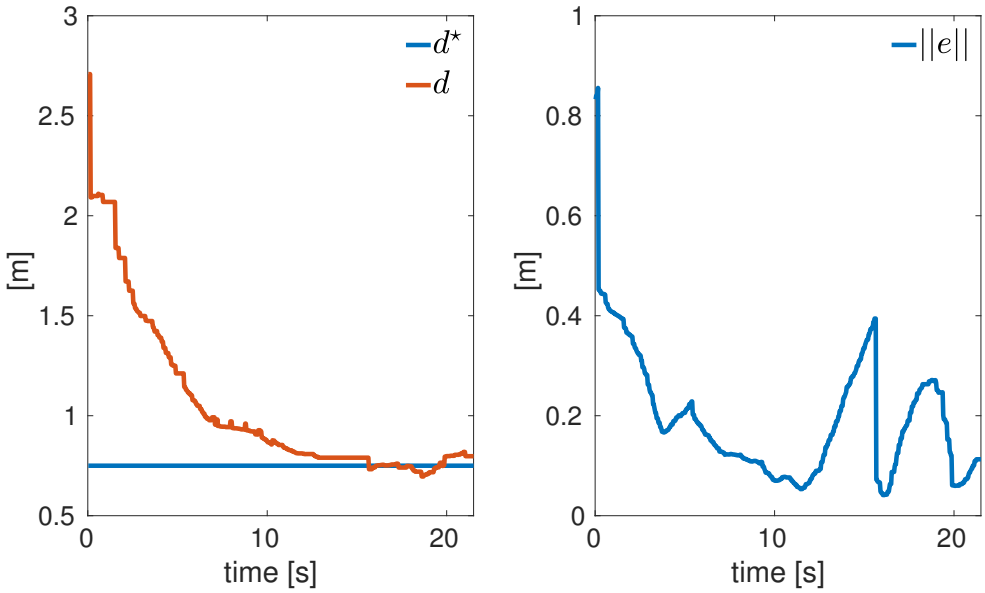


Figure 5-29: The figure shows on the left side the estimated distance (d red line) between quadrotor and ArUco marker computed using Eq. (3-9) with respect to the desired chosen value ($d^\star$ blue line). On the right side it is shown the evolution in time of the norm of the IBVS error given in Eq. (3-25). The result shows in this figure belongs to the mission task number 14 in which the quadrotor uses the bottom camera to approach a static object up to a desired distance (equal to 0.75m), minimizing the error ($|e|$) between the estimated and the desired marker's corners and center. When this distance is reached the quadrotor lands on the platform.

Chapter 6

Conclusions

In this thesis a quadrotor controller framework and a vision based planner have been developed. The explanation of how they have been designed it is provided respectively in chapter 2 and 3. The experiment chosen to validate them has been presented in chapter 4 and the associated results have been shown in 5. It is time now to derive the conclusion of this thesis and discuss what it has to be done in the future to improve the developed algorithms. It has been chosen to separate the conclusion associated to the navigation controller framework from the ones related to the vision based planner.

6-1 Navigation controller framework conclusions

The navigation controller framework has been designed to able to communicate with both Parrot AR. Drone 2.0 and Pixhawk autopilot. It is composed by five different modules (horizontal position controller, horizontal speed controller, vertical position controller, vertical speed controller, yaw controller) where a total of seven 2DOF PID controller allow the quadrotor to track simultaneously either the quadrotor's desired positions and the yaw angle ($x^{\star W}, y^{\star W}, z^{\star W}, \psi^{\star W}$) or the desired velocities and the yaw angle ($\dot{x}^{\star W}, \dot{y}^{\star W}, \dot{z}^{\star W}, \psi^{\star}$) or the horizontal desired velocities, the altitude and the yaw angle ($\dot{x}^{\star W}, \dot{y}^{\star W}, z^{\star W}, \psi^{\star W}$). The navigation controller conclusions are summarized here.

- The controller framework shown in Figure 2-1 has shown to be compatible with both AR Drone 2.0 and Pixahwk autopilot. Furthermore, the cascade design choice obtained designing different modules able to be started and stopped has permitted in the same mission to make the quadrotor firstly track the desired pose and secondly the translational velocities provided by the vision based planner. This has been possible simply stopped the horizontal and vertical position controller modules. The design has been inspired by the Simulink design where blocks (modules) running at a precise frequency are used to design controller's structures. The tuning of the controller's parameters has been done simplified the quadrotor autonomous navigation into subproblems where

the small angle angle approximation assumption has been found to be a satisfactory assumption to ensure indoor autonomous quadrotor navigation.

- The modularity of the design has enabled to easily replace the navigation controller modules with new modules with the same inputs and outputs and immediately test them running an autonomous mission. Thus it is possible to state that the modularity of the design has simplified the development and test of new controllers. This feature allows different people dealing with different control design to focus only on the development of the controller. Indeed a controller library is available where different controller can be added. This choice allows to easily select inside a module which controller available in the library better suits to the task problem.
- The simulation framework chosen combining the Gazebo simulation with the PX4 Software-In-The-Loop has allowed to validate the architecture design and the communication with different Aerostack modules such as EKF and Mission planner. Thanks to the simulation environment it has been possible to improve the design avoiding useless crashes. However in moving from simulation to real world environment it is required to perform a fine tuning of the controller's parameters.
- The design has shown to be robust to estimation errors introduced by the EKF. Indeed the real experiment has been conducted using EKF measurements with the aim to increase the level of autonomy of the quadrotor.
- The convention chosen for the navigation controller outputs, feedbacks and reference inputs data are in tune with the standard convention used in robotics to ensure that different state of art algorithms whose are developed using ROS can easily be combined with the navigation controller framework.

6-2 Vision based planner to approach either static or moving objects conclusion

It is made up by three different modules (perception, image state estimator, image based visual servo) whose final goal it is to calculate the desired translational velocities $(\dot{x}^{\star W}, \dot{y}^{\star W}, \dot{z}^{\star W})$ required by the quadrotor to approach up to a desired distance a chosen static or moving object keeping the detected center of the latter in a precise position on the current image plane. The vision based planner does not require to know neither the quadrotor's pose nor the visual marker one. The vision based planner conclusions are summarized here.

- The generalization of the task solved by the vision based planner which consists in approach to an object up to a desired distance has permitted to use the vision based planner to solve different tasks such as approach to a static or moving object or land on a static or moving platform.
- The division of the vision based planner problem into subproblems represented using modules has clearly underlined which are the three main components required by the vision based planner to solve the task. The latter are the detector, the marker's corners and center estimator and the image based visual servo controller located respectively

inside the perception, the image state estimator and the image based visual servo modules. This division has permitted to improve separately the three modules and test the overall planner in a fast and easy way. Indeed if a new detector it is designed it is only required to add it inside the perception module. Then it is required to compare the output of the already tested detector with the new developed. If the output are satisfactory it is possible to immediately run a fully autonomous mission using the new detector to accomplish the task. The same consideration can be done for both the marker's corner and center estimator and the control law used to derive the translational quadrotor velocities.

- The trajectory that the quadrotor performs in the real world when the vision based planner is used has been found to be strongly dependent on the choice of the controller parameter λ (Table 5-15 and Table 5-8). The latter are scaling translational velocities factors that can be used to modified the quadrotor trajectory in the real world. To be precise closer they are to the same value smother will appear the quadrotor trajectory in approaching to the marker.

- It has shown that it is able to work without knowing neither the object position in space nor the quadrotor one. Indeed the vision based planner solves the problem of how the quadrotor can move with respect to an object whose pose it is unknown and can change randomly. In term of quadrotor measurements it is required to only estimate the quadrotor velocities and yaw angle.

- The choice to make the vision based planner able to easily switch between front and bottom camera has permitted to the quadrotor to solve different tasks such as object following and autonomous landing in the same mission simply selecting the camera that the vision based planner has to use. This design has showed great advantages especially related to the simplification of the mission's design.

6-3 Future works

The presented designs have been proved that can be used to increase the level of autonomy of the quadrotor in performing autonomous missions. However, a great deal of improvements can be done related to both the navigation controller and the vision based planer designs.

6-3-1 Navigation controller framework future work

The navigation controller framework improvements are summarize here

- The navigation controller framework has been validated only in simulation environment for a quadrotor equipped with Pixhawk autopilot. Thus it is required to test it in a real quadrotor equipped with Pixhawk autopilot.

- To make the navigation controller as less dependent as possible from the choice of the autopilot it is required to test how it performs with different autopilots available on the market and ROS compatible.

- Identify a proper quadrotor model ensures a fast and accurate tuning of the 2DOF PID controller parameters. For this reason it is required to improve the navigation controller framework design using system identification techniques to derive and validate better quadrotor models that can be used to design new controllers or improve the tuning of the available ones.

6-3-2 Vision based planner future work

The vision based planner improvements are summarize here

- Improve the perception module design adding different type of detectors able to extract the side and the center of a given object in pixels coordinated. The constriction that it is only possible to approach an object on which a specific marker is applied on it has to be removed.
- Improve the image state estimator model (velocity constant model) with the aim to have a more reliable and robust estimation of the object corners' and center.
- Improve the image based visual servo control law such that it is able to deal with dynamic obstacles appearing between the camera and the chosen object.
- Remove the assumption that the marker is planar and use the marker orientation to also control the quadrotor rotation about the (z) axis.
- Validate how the $(2D)$ trajectory (on the image plane) of the markers' corner and center change in using different type of interaction matrices to exploit which among the desired, the estimate and the average better suits to a specific quadrotor task.
- Replace the bottom quadrotor camera with a Gimbal and use the image based visual servo controller outputs to control both the translational quadrotor velocities and the Gimbal angular velocities to avoid to loose the marker from the acquired image. This choice reduce the possibility that the object disappears from the image and it uses all the six image based visual servo controller outputs to control the translational quadrotor velocities and the the Gimbal angular velocities.
- Remove the assumption that the marker is planar in the estimation of the distance between the quadrotor and the chosen camera.
- Compare the image based visual servo design with both the position and homography based visual servo.
- Validate the maximum speed at which the quadrotor can follow a moving object or land on a moving platform using the vision based planner design in combination with the navigation controller framework.

Appendix A

Quadrotors

A-1 Parrot AR.Drone 2.0 (Parrot autopilot quadrotor)

Figure A-1: AR. Drone 2.0 with outdoor hull

It is a wireless quadrotor that can be controlled using the open source ROS Driver package *ardrone_autonomy* [1]. It is equipped with a front HD CMOS Video Camera: 720p (1280×720 pixels) at 30fps (92° wide angle lens) and a bottom VGA CMOS Video Camera: QVGA (320×240) at 60fps (64° wide angle lens). Both the cameras data are available but only one per time can broadcast camera image data. Thus, the user according to its purpose has to select from which camera get the image data. A resume of the AR. Drone 2.0 technical specification is shown in the following table.

[1] http://wiki.ros.org/ardrone_autonomy

Table A-1: Parrot AR.Drone 2.0 Technical Specification

	Ardrone 2.0 with outdoor hull
Weight	380 grams
Dimensions	$35(length) \times 28(width) \times 11(height)$ cm
Battery	Lithium polymer battery (3 cells, 11.1V,1500mAh) Running time: 12 minutes
Processor	1GHz 32 bit ARM Cortex A8 processor with 800MHz video DSP TMS320DMC64x
Operating System	Linux 2.6.32
Front Camera	Resolution:1280x720 pixels (720p) Video Rate 30fps 92° wide-angle diagonal lens camera, HD CMOS sensor
Vertical Camera	Resolution:320x240 pixels (QVGA) Video Rate 60fps 64° wide-angle diagonal lens camera, CMOS sensor
Ultrasound Altimeter	Emission frequency: 40kHz , Range: 6m
Accelerometer	3-axis, +/- 50mg precision
Gyroscope	3-axis, 2000°/second precision
Magnetometer	3 axis, 6° precision
Motors	4 brushless inrunner motors: 14.5 watts and 28500 rev/min
Connection	Wi-Fi b g n, b=(2.4GHz-11 Mb/s) g=(2.4GHz-54Mb/s) n=(2.4GHz-300Mb/s))
Price	249.99 Euro

A-2 Parrot Bebop 2.0 (Parrot autopilot quadrotor)

Figure A-2: Bebop 2.0

The Bebop 2 quadrotor also called AR. Drone 3.0 it is an evolution of the AR Drone 2.0. Comparing with the AR. Drone 2.0 it is a lighter and smaller quadrotor with higher flying

capability. Furthermore it is equipped with a 14 Mega-pixels CMOS front video camera: HD 1080p 1920 × 1080*p* (30 fps) Fisheye (180° wide angle lens) with digital stabilization implemented. A ROS Driver package called *bebop autonomy* is available making it compatible with ROS. The front camera can be controlled by software and the user can choose to look front or backward tilting the virtual camera by software command. Furthermore, thanks to the software image stabilization and the 14 Mega-pixel bottom side image view this quadrotor becomes a more suitable platform to test landing algorithm than the previous one (AR. Drone 2.0). However a low frequency (5 Hz) IMU data is available using the ROS driver package *bebop_autonomy*. This limitation makes difficult to estimate the quadrotor position and orientation in world using an EKF. To overcome this problem Parrot has developed a navigation system called S.L.A.M.dunk[2]. It is a 140 grams platform equipped with a Fish-eye stereo camera with a 1500 × 1500 resolution at 60fps, a Ubuntu computer with a ROS compatible SDK, IMU, magnetometer,ultrasound and barometer sensors. Thanks to the high resolution stereo camera the platform is able to generate an environment depth map used for both identify obstacles and measure how far they are from the quadrotor itself. The combination of the S.L.A.M.dunk with the Bebop 2 makes this quadrotor having all the sensors required to navigate autonomously in a GPS-denied environment and for this reason a relevant investigation research product. A resume of the Bebop 2.0 technical specification is shown in the following table.

Table A-2: Parrot Bebop 2.0 Technical Specification

	Bebop 2.0
Weight	500g
Dimensions	21 × 26 × 10 cm
Battery	Lithium polymer battery (11.1V, 2700mAh), 25 minutes fight time
Processor	ARM Cortex-A9, Dual core processor with quad-core GPU
Operating System	Linux
Front Camera	Resolution: 4096 × 3072 pixels (1080p) Video Rate 30fps, 3-axis digital system stabilization Sunny 180° wide-angle diagonal fish-eye lens: 1/2.3" aperture, HD CMOS 14 Mpx
Sensors	vertical stabilization camera (16fps), Ultrasound Altimeter, Pressure Sensor, 3-axis gyroscope, Accelerometer, 3-axis magnetometer, Global Navigation Satellite System (GNSS) chipset (GPS + GLONASS)
Motors	4 brushless motor, rotation speed is 7500 rpm in run-up and can reach over 12000 rpm.
Connection	Wi-Fi 802.11 a b g n ac
Price	500 Euro

[2] https://www.parrot.com/us/business-solutions/parrot-slamdunk

A-3 Eagle (Pixhawk autopilot quadrotor)

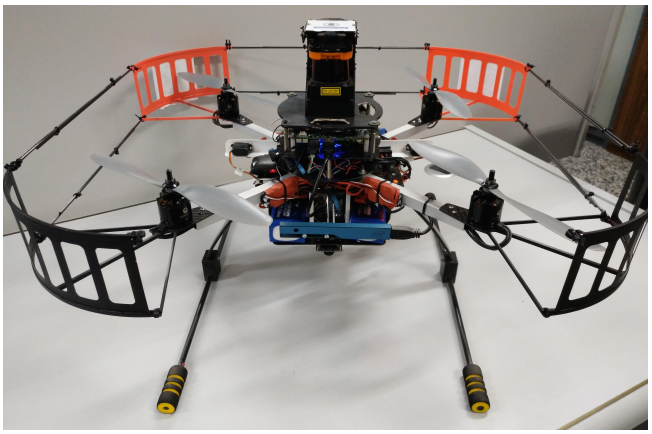


Figure A-3: Eagle with protection

Table A-3: Eagle Technical Specification

	Eagle with protection
Weight	3.2Kg, maximum payload capacity 1Kg
Dimensions	$61 \times 61 \times 40$ cm
Battery	2 Lithium polymer battery 4500mAh each, 12 minutes fight time
Processor	Intel-NUC 6i5SYK, 16GB of RAM and SSD de 256GB computer
Operating System	Ubuntu 14.04.5 LTS (Trusty Tahr) with ROS Jade and Aerostack installed
Autopilot	Pixhawk Autopilot of 3D Robotics, IMU and magnetometer integrated in the Autopilot, RC Module for user control
Sensors	Hokuyo Laser range finder UTM-30 LX (30m 270°), Intel Realsense RGB-D camera, RGB 180 degree fisheye lens bottom camera, Lightware altimeter SF10/A, 2 servos controlled using PWM signals from an oboard Arduino pro-mini board.
Oboard sensor communication	USB connection
Motor	4 motors of T-Motor MT2814-10 of 770KV with 2 blade propeller of 11×4.7 inch
Ground Station Computer to Onboard Computer communication	Wi-Fi
Price	6000 Euro

A-4 Sparrow (Pixhawk autopilot quadrotor)

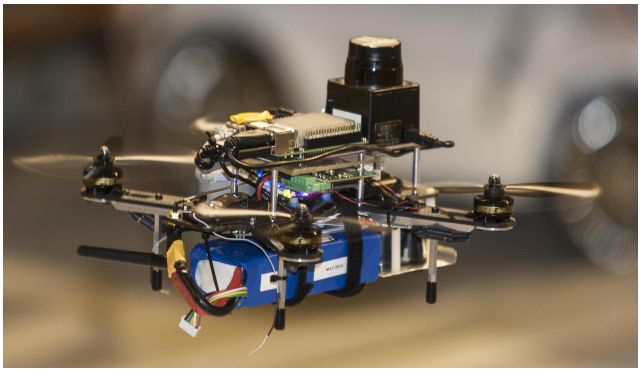


Figure A-4: Sparrow

Table A-4: Sparrow Technical Specification

	Sparrow
Weight	1.2Kg
Dimensions	21 × 24 × 19 cm
Battery	1 Lithium polymer battery 5000mAh each, 12 minutes fight time
Onboard Processor	UpBoard Intel Atom TM x5-Z8350, RAM 4GB DDR3L-1600, Storage 32GB eMMC, 4*USB2.0+USB3.0 OTG, alimentation 5VDC
Operating System	Ubuntu 16.04.2 LTS (Xenial Xerus) with ROS Kinetic and Aerostack installed
Autopilot	Pixhawk Mini Autopilot of 3D Robotics, IMU and magnetometer integrated in the Autopilot , RC Controller Module to user control
Sensors	Hokuyo URG-04LX-UG01 (5.6m, 240° view angle), Lightware SF10/A (up to 25m)
Oboard sensor communication	USB connection
Motor	4 motors of T-Motor F40 II 2400KV with 2 blade propeller
Ground Station Computer to Onboard Computer communication	Data Link, Module OEM IPnDDL, frequency 5.8GHz, up to 12Mbps data transfer, electromagnetic power up to 1W (range greater than 1 Kilometer), alimentation 7-30VDC
Price	4000 Euro

Appendix B

Navigation Controller Framework Appendix

B-1 Quadrotor dynamics

B-1-1 Translational dynamic

According to Euler's equation of motion and neglecting air drag forces the quadrotor translational dynamic can be described as

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -m \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + R_R^W \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \tag{B-1}$$

which means that the overall force applied to the quadrotor expressed in the world reference frame is given by the gravity force acting only along the z negative direction of the world frame plus the thrust force. This force is always oriented along the positive direction of the z axis of the robot frame. However due to the fact that the robot frame directions will change when the quadrotor moves, a rotation matrix able to transform a vector in the robot frame to the corresponding vector in the world frame is applied. Using the rotation matrix R_R^W it is possible to map the thrust vector force into the corresponding force applied in the three axis of the world frame. The resulting rotation matrix R_R^W is able to transform the orientation of a vector in the robot frame coordinates into the corresponding orientation of the same vector in the world frame coordinates is calculated using ZYX convention. The final rotation matrix describing how to transform a vector from robot frame coordinates to world ones is

the following

$$R_R^W = R_z(\psi)R_y(\theta)R_x(\phi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} =$$
$$= \begin{bmatrix} cos(\psi)cos(\theta) & cos(\psi)sin(\theta)sin(\phi) - sin(\psi)cos(\phi) & cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) \\ sin(\psi)cos(\theta) & sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) & sin(\psi)sin(\theta)cos(\phi) - cos(\psi)sin(\phi) \\ -sin(\theta) & cos(\theta)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \tag{B-2}$$

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \begin{bmatrix} ... & ... & cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) \\ ... & ... & sin(\psi)sin(\theta)cos(\phi) - cos(\psi)sin(\phi) \\ ... & ... & cos(\theta)cos(\phi) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \tag{B-3}$$

The final translational dynamic of the quadrotor isolating the acceleration results being

$$\begin{aligned} \ddot{x} &= \Big(cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) \Big) \frac{T}{m} \\ \ddot{y} &= \Big(sin(\psi)sin(\theta)cos(\phi) - cos(\psi)sin(\phi) \Big) \frac{T}{m} \\ \ddot{z} &= \Big(cos(\theta)cos(\phi) \Big) \frac{T}{m} - g \end{aligned} \tag{B-4}$$

The Maclaurin expansion (the Taylor expansion about 0) of the sine and cosine of angle η is

$$sin(\eta) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} \eta^{2n+1} = \eta - \frac{\eta^3}{3!} + \frac{\eta^5}{5!} - \frac{\eta^7}{7!} + \cdots = \eta - \frac{\eta^3}{6} + \frac{\eta^5}{120} - \frac{\eta^7}{5040} + \cdots$$

$$cos(\eta) = \frac{d}{d\eta} sin(\eta) = \frac{d}{d\eta} \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} \eta^{2n+1} = \frac{d}{d\eta} \Big(\eta - \frac{\eta^3}{3!} + \frac{\eta^5}{5!} - \frac{\eta^7}{7!} + \cdots \Big) =$$
$$= \frac{d}{d\eta} \Big(\eta - \frac{\eta^3}{6} + \frac{\eta^5}{120} - \frac{\eta^7}{5040} + \cdots \Big) = 1 - \frac{\eta^2}{2} + \frac{\eta^4}{24} - \frac{\eta^6}{720} + \cdots$$

Assuming small angle approximation and neglecting higher order term we write

$$\begin{aligned} sin(\eta) &\approx \eta \\ cos(\eta) &\approx 1 \end{aligned}$$

According to small angle approximation assumption the dynamics can be rewritten as

$$\begin{aligned} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} &= \frac{T}{m} \begin{bmatrix} cos(\psi) & sin(\psi) \\ sin(\psi) & -cos(\psi) \end{bmatrix} \begin{bmatrix} \theta \\ \phi \end{bmatrix} \\ \ddot{z} &= \frac{T}{m} - g \end{aligned}$$

B-1-2 Rotational dynamic

Looking at the rotational dynamic recalling Euler momenta equation of motion it is possible to write the rotational dynamic of the quadrotor as

$$I\dot{w} + w \times Iw = \tau \tag{B-5}$$

where the upper equations describes how the rotational dynamic changes in the world frame which it stands for how the angular acceleration change (or momenta) when some external torques τ are applied to the quadrotor. The inertial matrix I is a diagonal matrix thanks to the assumption that the rotating robot reference frame has its axis fixed to the body and parallel to the body's principal axis of inertia.

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{B-6}$$

To write the angular velocities as a function of the Euler angles the derivate of the rotation matrix able to transform a vector from the body to the world frame required. The latter is given by

$$\dot{R}_R^W = R_R^W \begin{bmatrix} w \end{bmatrix}_\times \tag{B-7}$$

where $\begin{bmatrix} w \end{bmatrix}_\times$ stands for the skew symmetric matrix resulting from vector $w \in \mathfrak{Re}^{3\times 1}$ and is computed as follows

$$\begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}_\times = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix} \tag{B-8}$$

The relation between angular velocities $w = \begin{bmatrix} w_x & w_y & w_z \end{bmatrix}^T$ with the derivative of pitch,roll and yaw $\dot{\alpha} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}$ is obtained

$$\begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} R_R^{WT} \dot{R}_R^W \end{bmatrix}^v = \begin{bmatrix} 1 & 0 & -sin(\theta) \\ 0 & cos(\phi) & cos(\theta)sin(\phi) \\ 0 & -sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = J(\alpha)\dot{\alpha} \tag{B-9}$$

where $\begin{bmatrix} R_R^{WT} \dot{R}_R^W \end{bmatrix}^v$ stands for the inverse operation of the skew symmetric one. Assuming small angle approximation the $J(\alpha)$ matrix relating angular velocities $w = \begin{bmatrix} w_x & w_y & w_z \end{bmatrix}^T$ with the derivative of pitch,roll and yaw $\dot{\alpha} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}$ results being the identity

$$J(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B-10}$$

Replacing Eq. (B-9) into Eq. (B-5) the rotational quadrotor dynamic equation becomes

$$IJ\ddot{\alpha} + I\dot{J}\dot{\alpha} + J\dot{\alpha} \times IJ\dot{\alpha} = \tau \tag{B-11}$$

where knowing that $J(\alpha)$ is the identity from equation Eq. (B-10) it is possible to state that

$$I\ddot{\alpha} = -\dot{\alpha} \times I\dot{\alpha} + \tau \rightarrow I\ddot{\alpha} = -\left[\dot{\alpha}\right]_{\times} I\dot{\alpha} + \tau \tag{B-12}$$

where $\left[\dot{\alpha}\right]_{\times}$ stands for the skew symmetric matrix resulting from vector $\dot{\alpha}$

$$\left[\dot{\alpha}\right]_{\times} = \begin{bmatrix} 0 & -\dot{\alpha}_3 & \dot{\alpha}_2 \\ \dot{\alpha}_3 & 0 & -\dot{\alpha}_1 \\ -\dot{\alpha}_2 & \dot{\alpha}_1 & 0 \end{bmatrix} \tag{B-13}$$

The complete rotational dynamics equations are

$$\begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = - \begin{bmatrix} 0 & -\dot{\psi} & \dot{\theta} \\ \dot{\psi} & 0 & -\dot{\phi} \\ -\dot{\theta} & \dot{\phi} & 0 \end{bmatrix} \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{B-14}$$

Isolating the angular acceleration $\begin{bmatrix} \ddot{\phi} & \ddot{\theta} & \ddot{\psi} \end{bmatrix}$ and writing the torques as (τ_x, τ_y, τ_z)

$$\begin{aligned} \ddot{\phi} &= \dot{\theta}\dot{\psi}\frac{I_y - I_z}{I_x} + \frac{\tau_x}{I_x} \\ \ddot{\theta} &= \dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} + \frac{\tau_y}{I_y} \\ \ddot{\psi} &= \dot{\phi}\dot{\theta}\frac{I_x - I_y}{I_z} + \frac{\tau_z}{I_z} \end{aligned} \tag{B-15}$$

Assuming that the derivative of the Euler angle are really small the rotational dynamic turns being

$$\begin{aligned} \ddot{\phi} &= \frac{\tau_x}{I_x} \\ \ddot{\theta} &= \frac{\tau_y}{I_y} \\ \ddot{\psi} &= \frac{\tau_z}{I_z} \end{aligned} \tag{B-16}$$

The relation between the torques (τ_x, τ_y, τ_z) and the angular speed of each motor (w_1, w_2, w_3, w_4) is derived as following.

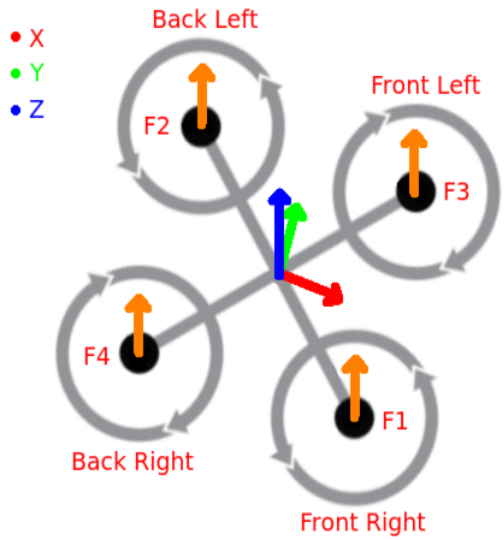


Figure B-1: X Configuration, motor number and body frame

$$\begin{aligned} T &= F_1 + F_2 + F_3 + F_4 = b(w_1^2 + w_2^2 + w_3^2 + w_4^2) \\ \tau_x &= F_3 + F_2 - F_1 - F_4 = Lb(w_3^2 + w_2^2 - w_1^2 - w_4^2) \\ \tau_y &= F_2 + F_4 - F_3 - F_1 = Lb(w_2^2 + w_4^2 - w_3^2 - w_1^2) \\ \tau_z &= F_1 + F_2 - F_4 - F_3 = db(w_1^2 + w_2^2 - w_4^2 - w_3^2) \end{aligned} \tag{B-17}$$

$$\begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ -Lb & Lb & Lb & -Lb \\ -Lb & Lb & -Lb & Lb \\ db & db & -db & -db \end{bmatrix} \begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{bmatrix} = A \begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{bmatrix} \tag{B-18}$$

$$det(A) = 16L^2b^4d \quad \neq 0 \quad \forall b, d, L \neq 0 \tag{B-19}$$

$$\begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4b} & \frac{-1}{4Lb} & \frac{-1}{4Lb} & \frac{1}{4bd} \\ \frac{1}{4b} & \frac{1}{4Lb} & \frac{1}{4Lb} & \frac{1}{4bd} \\ \frac{1}{4b} & \frac{1}{4Lb} & \frac{-1}{4Lb} & \frac{-1}{4bd} \\ \frac{1}{4b} & \frac{-1}{4Lb} & \frac{1}{4Lb} & \frac{-1}{4bd} \end{bmatrix} \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{B-20}$$

$$\begin{aligned} w_1 &= \sqrt{\frac{1}{4b}\left(T - \frac{\tau_x}{L} - \frac{\tau_y}{L} + \frac{\tau_z}{d}\right)} \\ w_2 &= \sqrt{\frac{1}{4b}\left(T + \frac{\tau_x}{L} + \frac{\tau_y}{L} + \frac{\tau_z}{d}\right)} \\ w_3 &= \sqrt{\frac{1}{4b}\left(T + \frac{\tau_x}{L} - \frac{\tau_y}{L} - \frac{\tau_z}{d}\right)} \\ w_4 &= \sqrt{\frac{1}{4b}\left(T - \frac{\tau_x}{L} + \frac{\tau_y}{L} - \frac{\tau_z}{d}\right)} \end{aligned} \tag{B-21}$$

where b , d and L are respectively the motor constant, the motor drag coefficient and the arm length whereas w_1, w_2, w_3, w_4 are the quadrotor angular velocity of each motor.

B-1-3 Simplified quadrotor model

The simplified quadrotor dynamics derived under the small angle approximation is the following

$$\begin{aligned}
\ddot{x} &= \frac{T}{m}\begin{bmatrix} cos(\psi) & sin(\psi) \end{bmatrix}\begin{bmatrix} \theta \\ \phi \end{bmatrix} \\
\ddot{y} &= \frac{T}{m}\begin{bmatrix} sin(\psi) & -cos(\psi) \end{bmatrix}\begin{bmatrix} \theta \\ \phi \end{bmatrix} \\
\ddot{z} &= \frac{T}{m} - g \\
\ddot{\phi} &= \frac{\tau_x}{I_x} \\
\ddot{\theta} &= \frac{\tau_y}{I_y} \\
\ddot{\psi} &= \frac{\tau_z}{I_z}
\end{aligned} \tag{B-22}$$

B-2 Navigation controller framework controller tuning experiment results

B-2-1 Multiples velocity $\dot{x}^{\star W}$ steps

This experiment has been done to validate how the $(2dofPid_{(\dot{x}^{\star},\hat{\dot{x}})\rightarrow\theta_v})$ performs in tracking multiples velocity $(\dot{x}^{\star})$ steps between -1 and 1 meters per seconds. The control parameters used in this experiment are given in Table 5-11. A video showing the experiment it is available here (https://www.youtube.com/watch?v=5Q4DxjXFzJE).

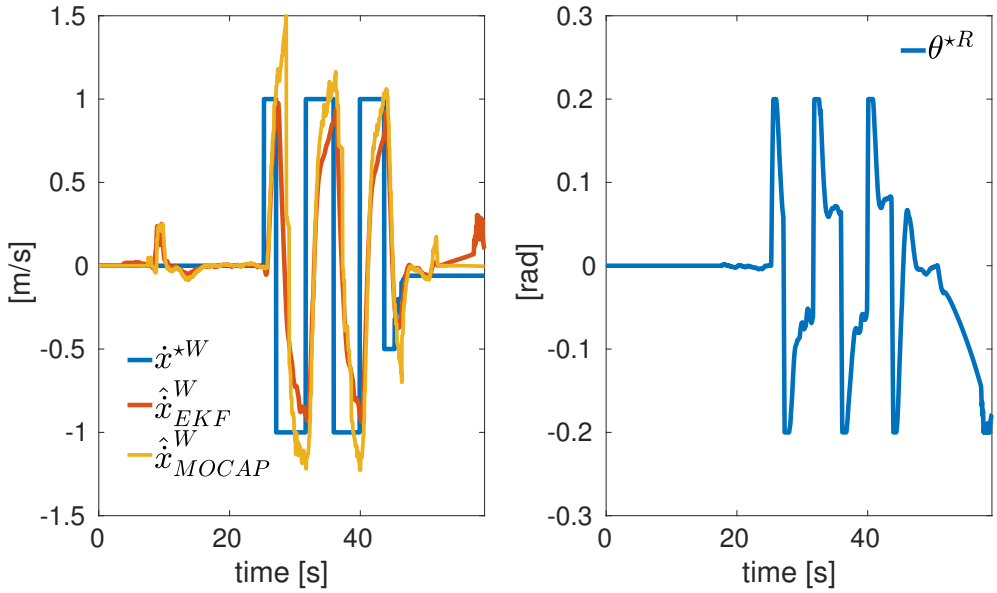


Figure B-2: The figure shown on the left side the reference velocity ($\dot{x}^{\star W}$) (blue line), the EKF measured one ($\hat{\dot{x}}^{W}_{EKF}$) (red line) and the ground truth ($\hat{\dot{x}}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the pitch ($\theta^{\star R}$) controller output calculated by the 2DOF PID controller ($2dofPid_{(\dot{x}^{\star},\hat{\dot{x}})\rightarrow\theta_v}$).

B-2-2 Multiples velocity $\dot{y}^{\star W}$ steps

This experiment has been done to validate how the ($2dofPid_{(\dot{y}^{\star},\hat{\dot{y}})\rightarrow\phi_v}$) performs in tracking multiples velocity ($\dot{y}^{\star}$) steps between -1 and 1 meter per seconds. The control parameters used in this experiment are given in Table 5-11. A video showing the experiment it is available here (https://www.youtube.com/watch?v=pHfy-lsMHYk).

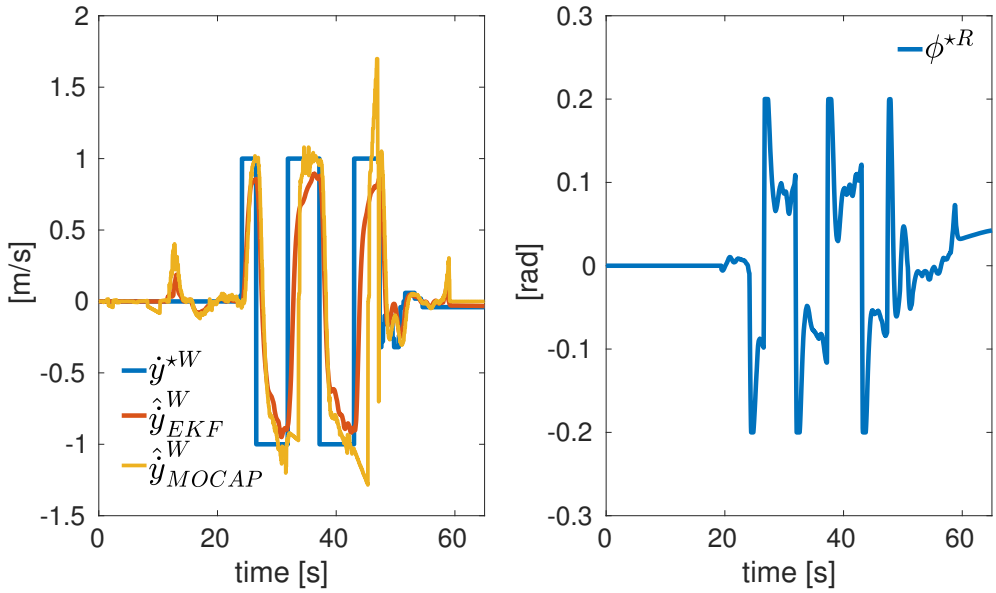


Figure B-3: The figure shown on the left side the reference velocity ($\dot{y}^{\star W}$) (blue line), the EKF measured one ($\hat{\dot{y}}^{W}_{EKF}$) (red line) and the ground truth ($\hat{\dot{y}}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the ($\phi^{\star R}$) controller output calculated by the 2DOF PID controller ($2dofPid_{(\dot{y}^{\star},\hat{\dot{y}})\rightarrow\phi_v}$).

B-2-3 Multiples position $x^{\star W}$ steps

This experiment has been done to validate how the ($2dofPid_{(x^{\star},\hat{x})\rightarrow\dot{x}^{\star}}$) performs in tracking multiples position ($x^{\star W}$) steps. The control parameters used in this experiment are given in Table 5-9. A video showing the experiment it is available here (https://www.youtube.com/watch?v=P6tBmfbejFg).

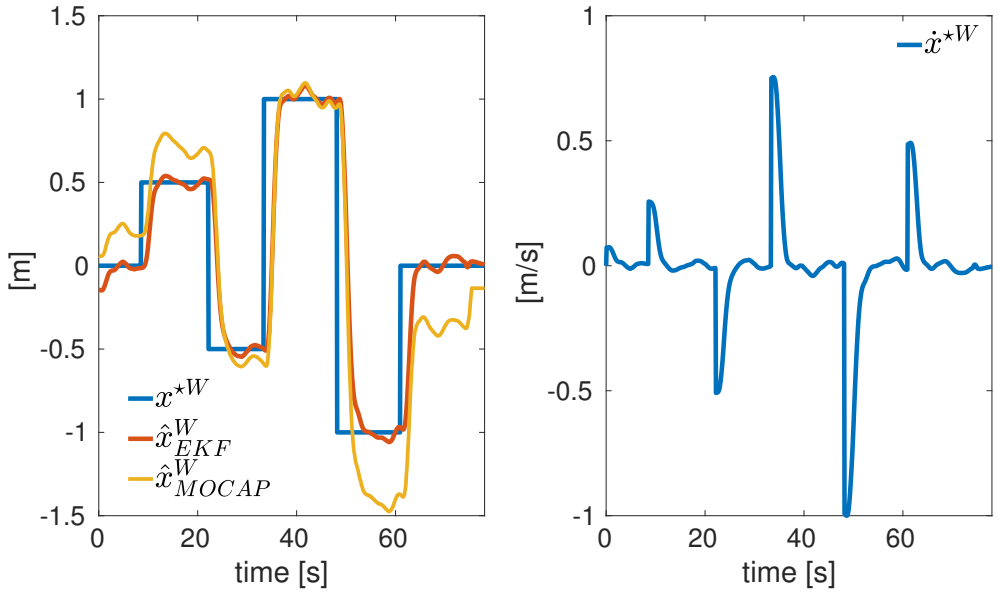


Figure B-4: The figure shown on the left side the reference position ($x^{\star W}$) (blue line), the EKF measured one ($\hat{x}^{W}_{EKF}$) (red line) and the ground truth ($\hat{x}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the velocity ($\dot{x}^{\star W}$) which is the controller output calculated by the 2DOF PID controller ($2dofPid_{(x^{\star},\hat{x})\rightarrow\dot{x}^{\star}}$).

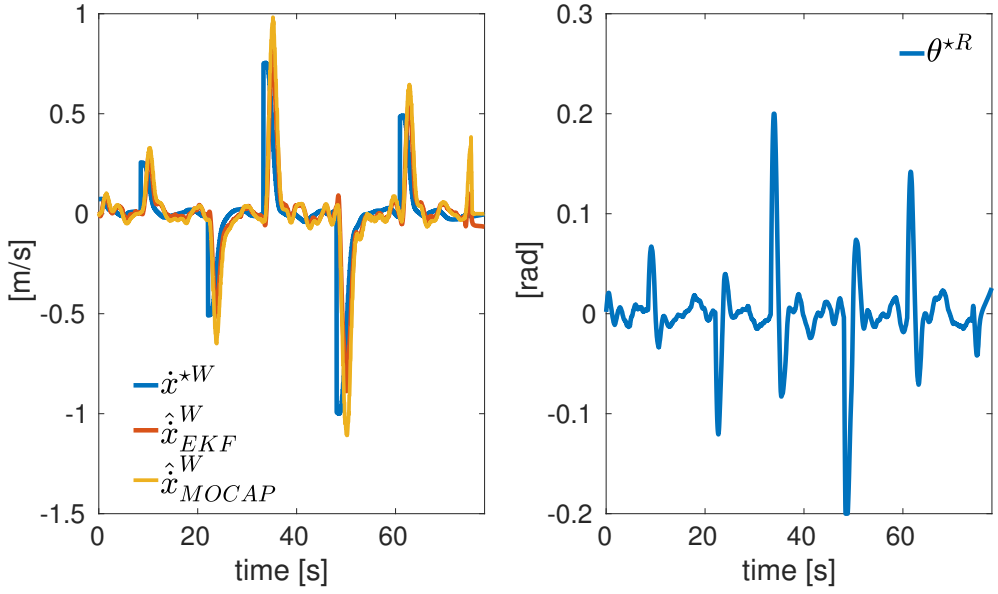


Figure B-5: The figure shown on the left side the reference velocity ($\dot{x}^{\star W}$) (blue line), the EKF measured one ($\hat{\dot{x}}^{W}_{EKF}$) (red line) and the ground truth ($\hat{\dot{x}}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the pitch ($\theta^{\star R}$) controller output calculated by the 2DOF PID controller ($2dofPid_{(\dot{x}^{\star},\hat{\dot{x}})\rightarrow\theta_v}$)

B-2-4 Multiples position $y^{\star W}$ steps

This experiment has been done to validate how the $(2dofPid_{(y^{\star},\hat{y})\rightarrow\dot{y}^{\star}})$ performs in tracking multiples position $(y^{\star W})$ steps. The control parameters used in this experiment are given in Table 5-9. A video showing the experiment it is available here (https://www.youtube.com/watch?v=2qNIbvRdwcE).

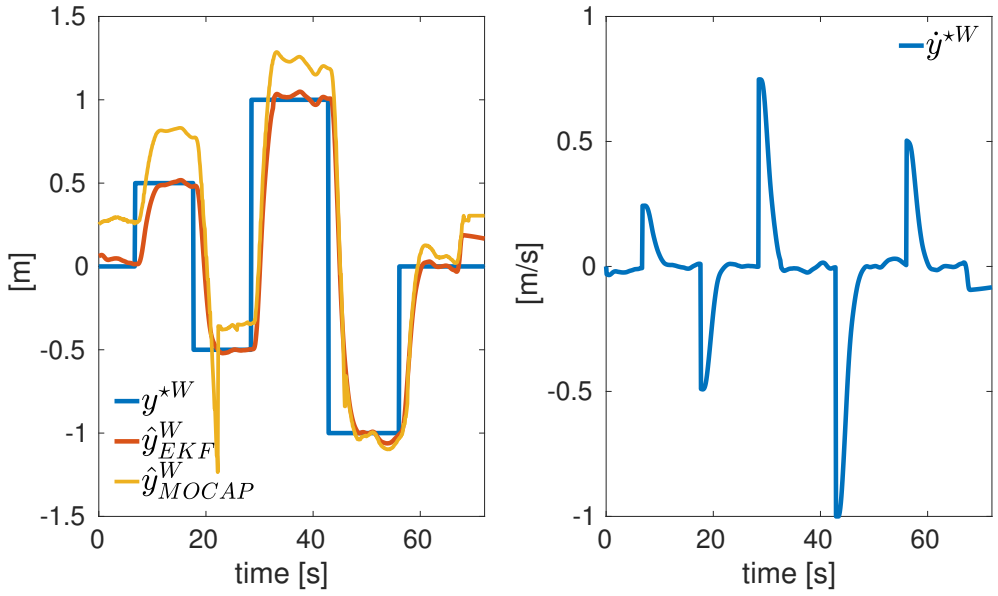


Figure B-6: The figure shown on the left side the reference position $(y^{\star W})$ (blue line), the EKF measured one $(\hat{y}^{W}_{EKF})$ (red line) and the ground truth $(\hat{y}^{W}_{MOCAP})$ (yellow line) available for comparison. On th right side it is shown the $(\dot{y}^{\star W})$ which is the controller output calculated by the 2DOF PID controller $(2dofPid_{(y^{\star},\hat{x})\rightarrow\dot{y}^{\star}})$.

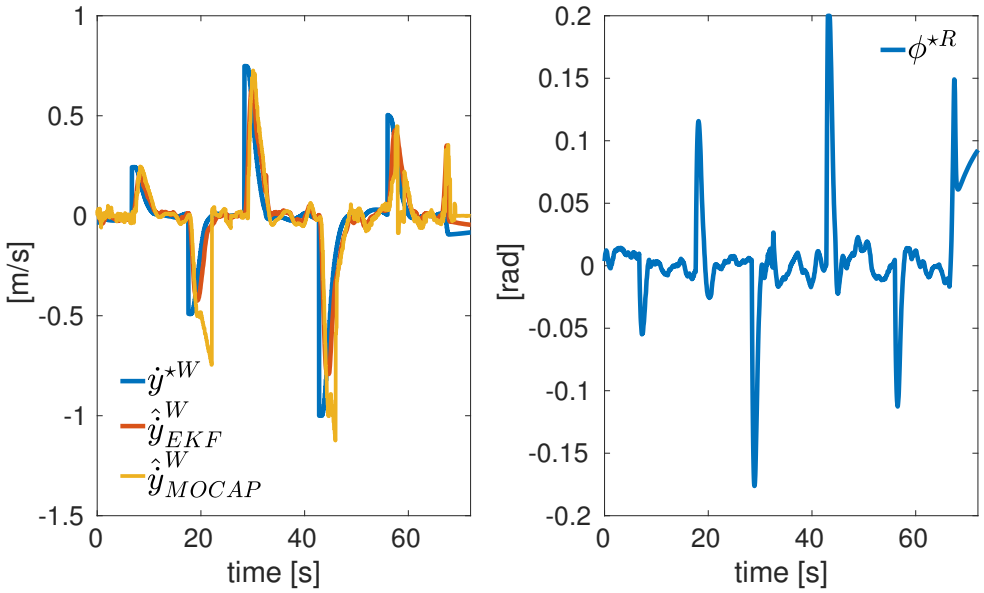


Figure B-7: The figure shown on the left side the reference velocity ($\dot{y}^{\star W}$) (blue line), the EKF measured one ($\hat{\dot{y}}^{W}_{EKF}$) (red line) and the ground truth ($\hat{\dot{y}}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the roll ($\phi^{\star R}$) controller output calculated by the 2DOF PID controller ($2dofPid_{(\dot{y}^{\star},\hat{\dot{y}})\rightarrow\phi_v}$)

B-2-5 Multiples position $z^{\star W}$ steps

This experiment has been done to validate how the ($2dofPid_{(z^{\star},\hat{z})\rightarrow\dot{z}^{\star}}$) performs in tracking multiples position ($z^{\star W}$) steps. The control parameters used in this experiment are given in Table 5-10. A video showing the experiment it is available here (https://www.youtube.com/watch?v=mTIoTSK9JLE).

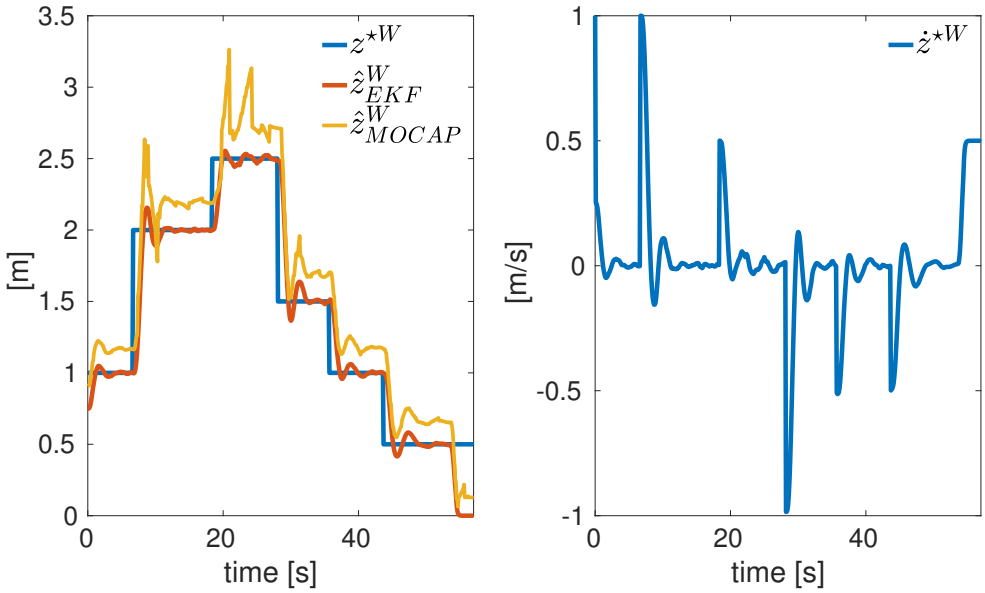


Figure B-8: The figure shown on the left side the reference position ($z^{\star W}$) (blue line), the EKF measured one ($\hat{z}^{W}_{EKF}$) (red line) and the ground truth ($\hat{z}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the velocity ($\dot{z}^{\star W}$) which is the controller output calculated by the 2DOF PID controller ($2dofPid_{(z^{\star},\hat{z})\rightarrow\dot{z}^{\star}}$).

B-2-6 Multiple yaw $\psi^{\star W}$ steps

This experiment has been done to validate how the ($2dofPid_{(\psi^{\star},\hat{\psi})\rightarrow\dot{\psi}^{\star}}$) performs in tracking multiples desird yaw ($\psi^{\star W}$) steps. The control parameters used in this experiment are given in Table 5-12. A video showing the experiment it is available here (https://www.youtube.com/watch?v=riIPh0GbR0k).

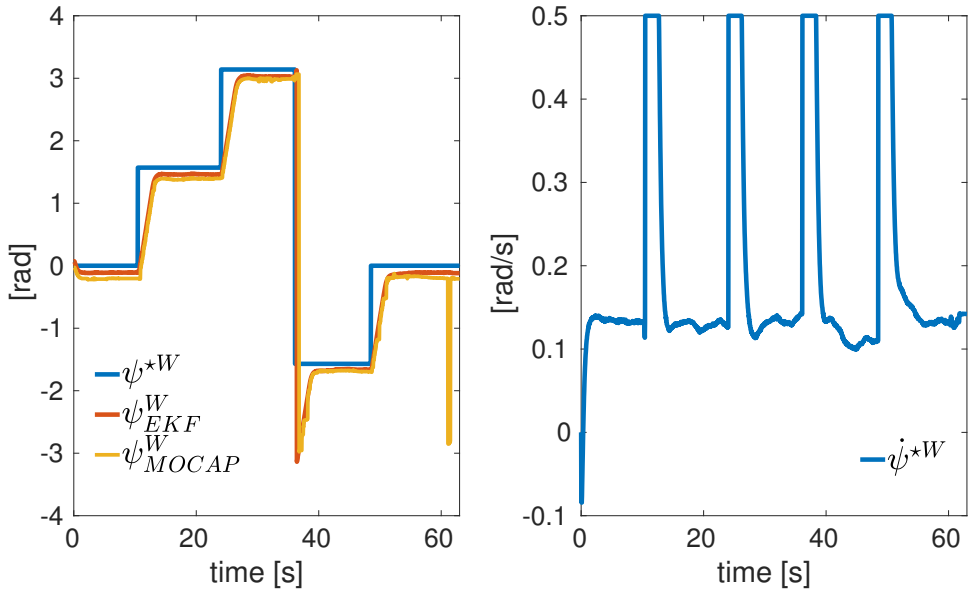


Figure B-9: The figure shown on the left side the reference yaw ($\psi^{\star W}$) (blue line), the EKF measured one ($\hat{\psi}^{W}_{EKF}$) (red line) and the ground truth ($\hat{\psi}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the yaw rate ($\dot{\psi}^{\star}$) which is the controller output calculated by the 2DOF PID controller ($2dofPid_{(\psi^{\star},\hat{\psi})\rightarrow\dot{\psi}^{\star}}$).

B-2-7 Navigation controller + EKF mission experiment

In this experiment the previously tuned controller are combined to solve a mission in which different desired poses have to be tracked. In performing the mission the Aerostack EKF state estimator is used. A video showing the experiment it is available here (https://www.youtube.com/watch?v=MsWUhc1AMzU).

Mission

1. *Take off*
2. *Go to point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 0, 0, 1, 0 \end{bmatrix}$
3. *Go to point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 1, 0, 1, 90 \end{bmatrix}$
4. *Go to point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 1, 1, 1, 180 \end{bmatrix}$
5. *Go to point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} -1, 1, 1, -90 \end{bmatrix}$
6. *Go to point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} -1, 0, 1, 0 \end{bmatrix}$
7. *Go to point* $\rightarrow \begin{bmatrix} x^{\star W} & y^{\star W} & z^{\star W} & \psi^{\star W} \end{bmatrix} = \begin{bmatrix} 0, 0, 1, 0 \end{bmatrix}$
8. *Land*

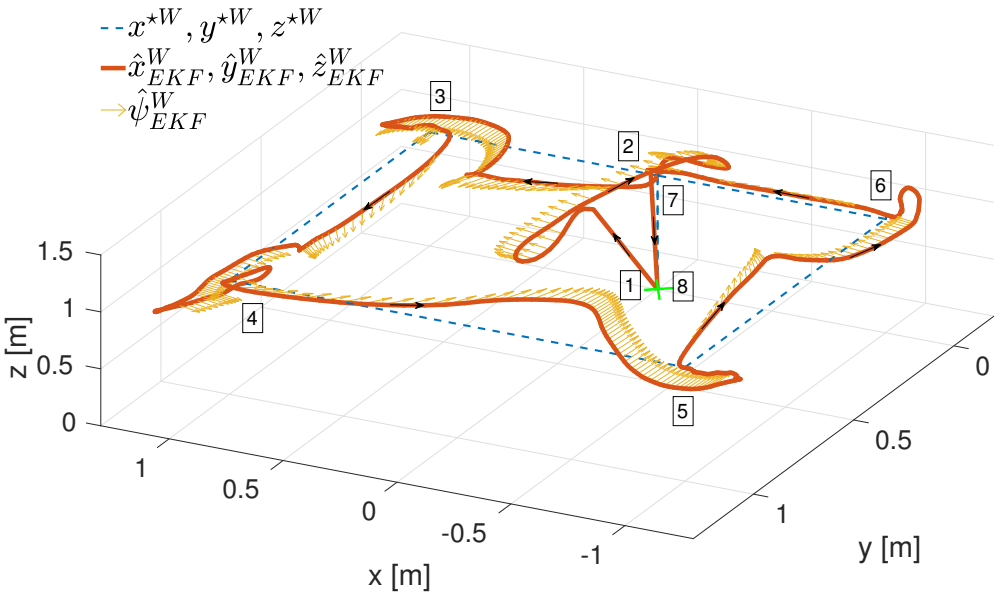


Figure B-10: The figure shows the quadrotor's $(3D)$ trajectory associated to the quadrotor's mission B-2-7. The EKF data are used to estimate the quadrotor's states (position,velocity,acceleration,orientation) expressed in world coordinate frame (red line). The numbers appearing on the figure are used to indicate the mission task's number that the quadrotor is facing.

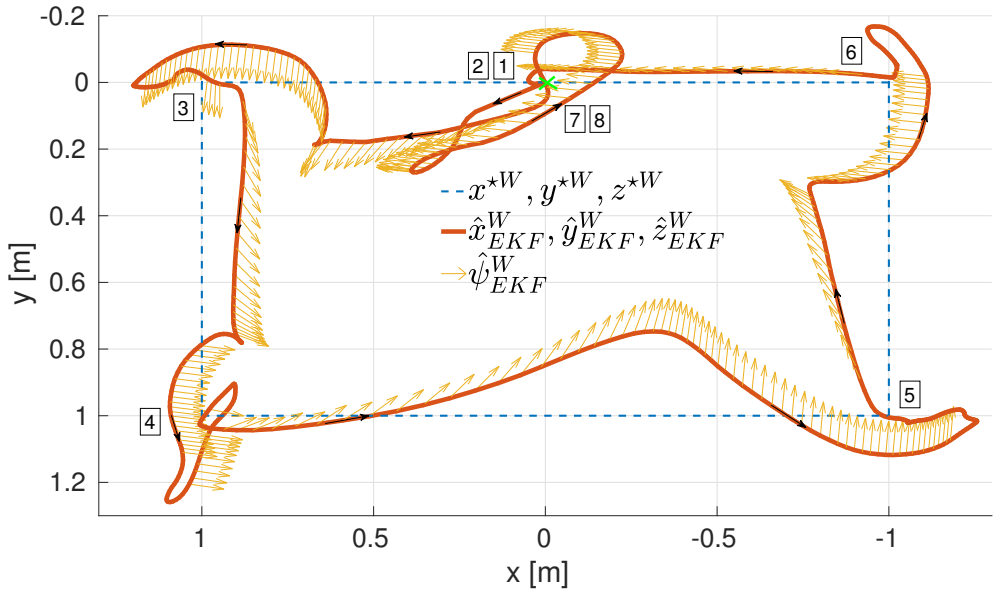


Figure B-11: The figure shows the $(2D)$ quadrotor's trajectory associated to the quadrotor mission B-2-7 which it stands for how the quadrotor moves along the (x) and (y) direction of the world coordinate frame. The numbers appearing on the figure are used to indicate the mission task's number that the quadrotor is facing.

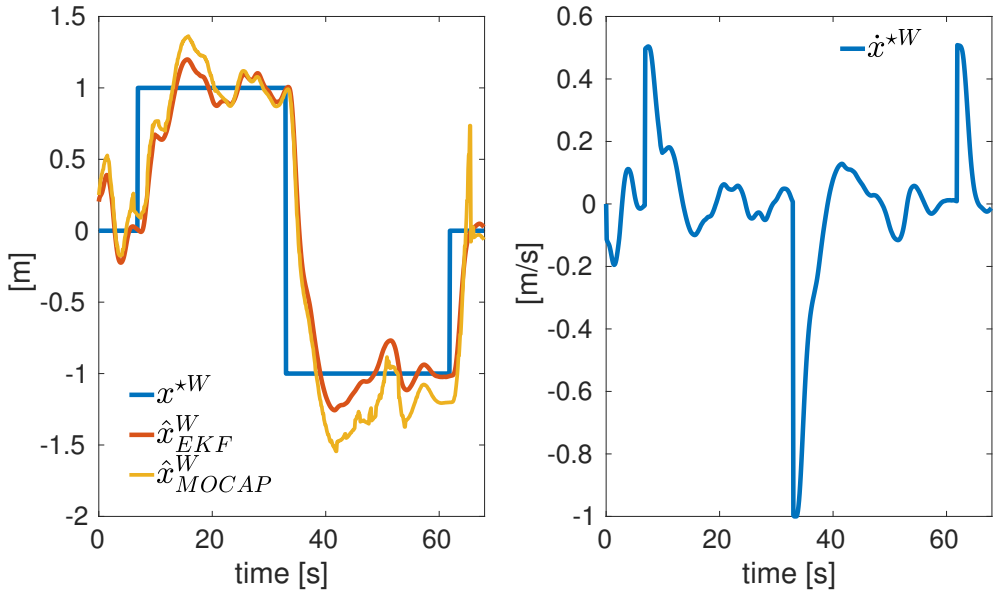


Figure B-12: The figure shown on the left side the reference position $(x^{\star W})$ (blue line), the EKF measured one $(\hat{x}^W_{EKF})$ (red line) and the ground truth $(\hat{x}^W_{MOCAP})$ (yellow line) available for comparison. On th right side it is shown the velocity $(\dot{x}^{\star W})$ which is the controller output calculated by the 2DOF PID controller $(2dofPid_{(x^\star,\hat{x})\rightarrow\dot{x}^\star})$.

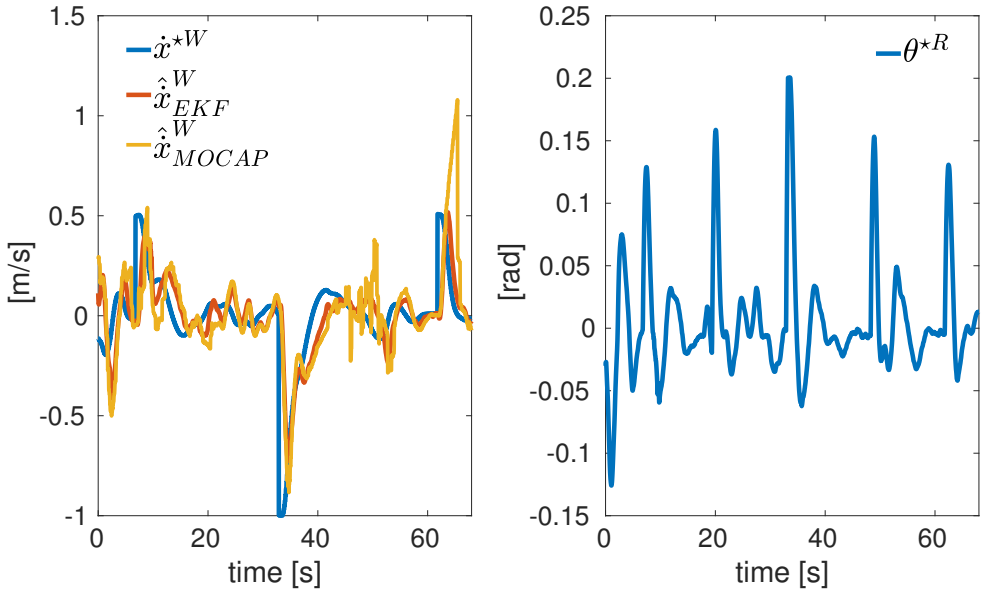


Figure B-13: The figure shown on the left side the reference velocity $(\dot{x}^{\star W})$ (blue line), the EKF measured one $(\hat{\dot{x}}^{W}_{EKF})$ (red line) and the ground truth $(\hat{\dot{x}}^{W}_{MOCAP})$ (yellow line) available for comparison. On th right side it is shown the pitch $(\theta^{\star R})$ controller output calculated by the 2DOF PID controller $(2dofPid_{(\dot{x}^{\star},\hat{\dot{x}})\rightarrow\theta_v})$

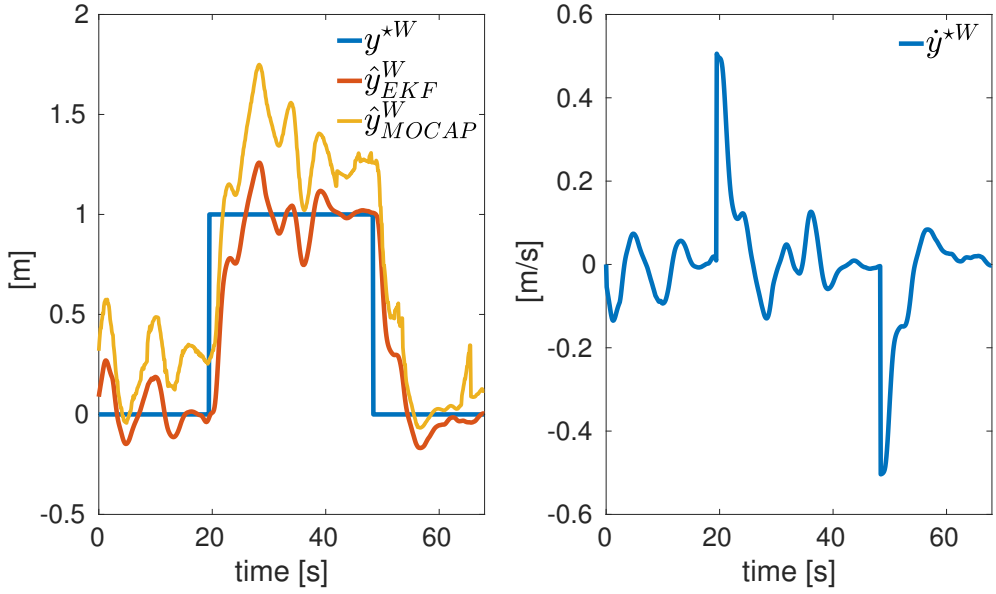


Figure B-14: The figure shown on the left side the reference position $(y^{\star W})$ (blue line), the EKF measured one $(\hat{y}^{W}_{EKF})$ (red line) and the ground truth $(\hat{y}^{W}_{MOCAP})$ (yellow line) available for comparison. On th right side it is shown the $(\dot{y}^{\star W})$ which is the controller output calculated by the 2DOF PID controller $(2dofPid_{(y^{\star},\hat{x})\rightarrow\dot{y}^{\star}})$.

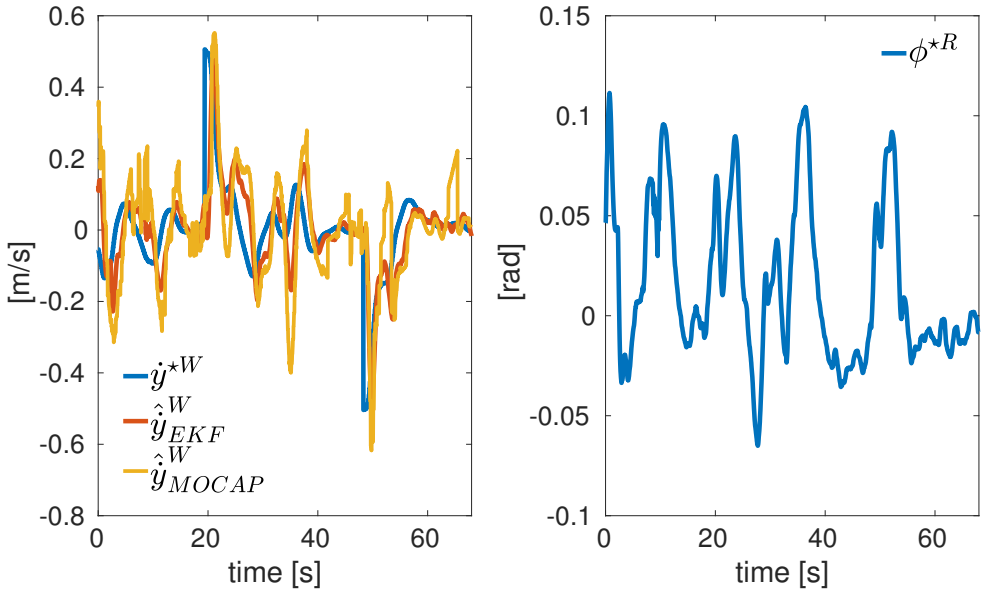


Figure B-15: The figure shown on the left side the reference velocity ($\dot{y}^{\star W}$) (blue line), the EKF measured one ($\hat{\dot{y}}^{W}_{EKF}$) (red line) and the ground truth ($\hat{\dot{y}}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the roll ($\phi^{\star R}$) controller output calculated by the 2DOF PID controller ($2dofPid_{(\dot{y}^{\star},\hat{\dot{y}})\rightarrow\phi_v}$)

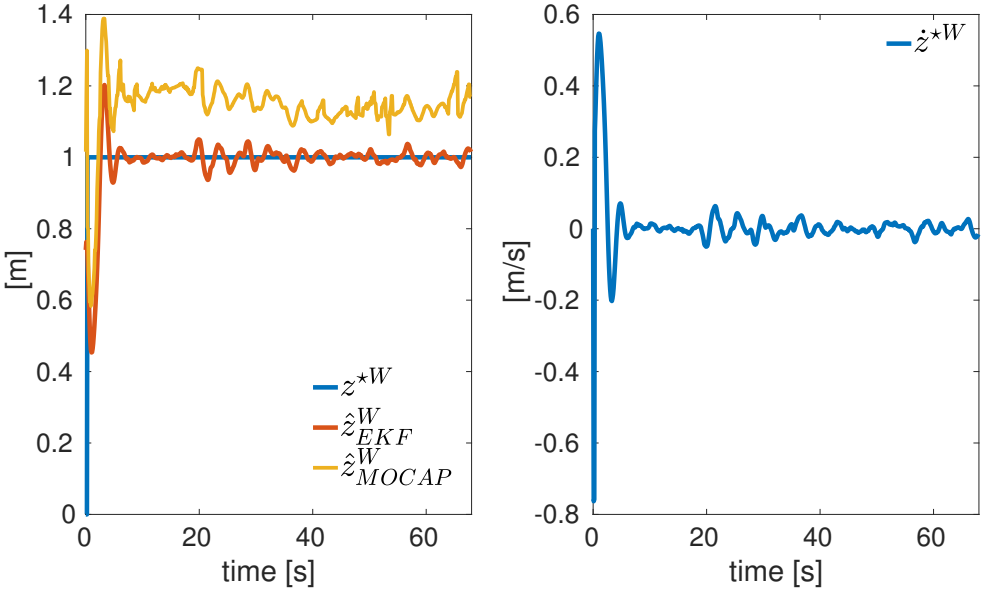


Figure B-16: The figure shown on the left side the reference position ($z^{\star W}$) (blue line), the EKF measured one ($\hat{z}^{W}_{EKF}$) (red line) and the ground truth ($\hat{z}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the velocity ($\dot{z}^{\star W}$) which is the controller output calculated by the 2DOF PID controller ($2dofPid_{(z^{\star},\hat{z})\rightarrow\dot{z}^{\star}}$).

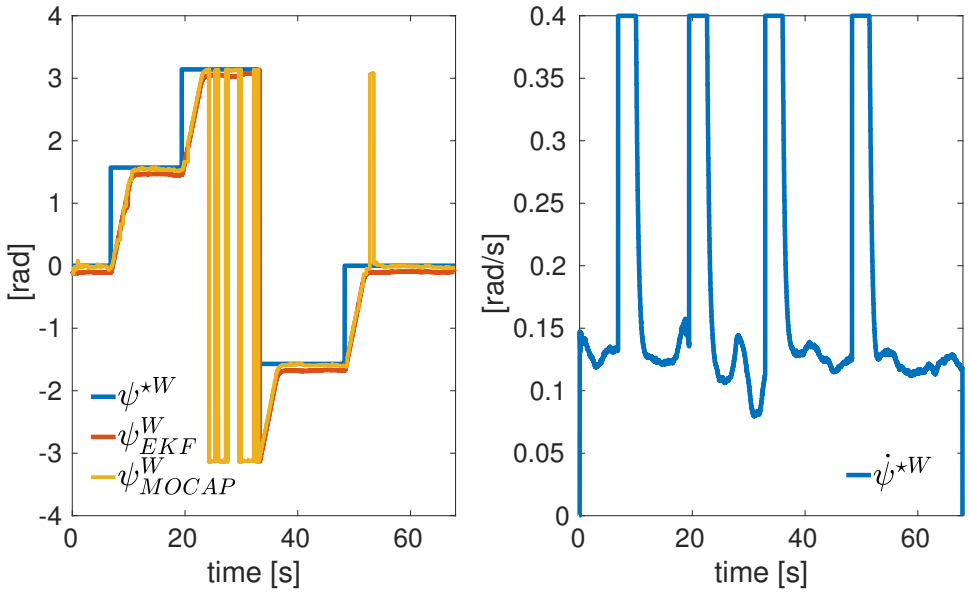


Figure B-17: The figure shown on the left side the reference yaw ($\psi^{\star W}$) (blue line), the EKF measured one ($\hat{\psi}^{W}_{EKF}$) (red line) and the ground truth ($\hat{\psi}^{W}_{MOCAP}$) (yellow line) available for comparison. On th right side it is shown the yaw rate ($\dot{\psi}^{\star}$) which is the controller output calculated by the 2DOF PID controller ($2dofPid_{(\psi^{\star},\hat{\psi})\rightarrow\dot{\psi}^{\star}}$).

B-3 Code used to send Pixhawk autopilot commands using the mavros package

An example of code used to send pitch, roll, yaw and thrust commands (${}^{PI}\theta^{\star R}, {}^{PI}\phi^{\star R}, {}^{PI}T^{\star R}, {}^{PI}\psi^{\star R}$) derived in Eq. (4-7) through the *mavros* topic *mavros/setpoint_raw/attitude* using as a feedback only the IMU data coming from */mavros/imu/data* is provided.

```
1     /*
2     1) Get IMU msg.orientation
3     from /mavros/imu/data
4     message type exmaple sensor_msgs::Imu msg;
5     */
6
7     /*
8     2) Extract yaw_IMU from IMU data in ENU (+ccw,-cw) [-pi pi]
9     */
10
11    //convert quaternion msg to eigen
12    Eigen::Quaterniond quaterniond;
13    tf::quaternionMsgToEigen(ImuMsgs.orientation, quaterniond);
14
15    //Converts ENU (Mavros) frame to NED  frame
16    //Rotating the frame in x-axis by 180 degrees
17    Eigen::Quaterniond BASE_LINK_TO_AIRCRAFT = mavros::UAS::
          quaternion_from_rpy(M_PI, 0.0, 0.0);
```

```
18      quaterniond = quaterniond*BASE_LINK_TO_AIRCRAFT;
19      //Rotating the frame in x-axis by 180 deg and in z-axis by 90 axis
20      Eigen::Quaterniond ENU_TO_NED = mavros::UAS::quaternion_from_rpy(M_PI
            , 0.0, M_PI_2);
21      quaterniond = ENU_TO_NED*quaterniond;
22
23      //converting back quaternion from eigen to msg
24      geometry_msgs::Quaternion quaternion;
25      tf::quaternionEigenToMsg(quaterniond, quaternion);
26      tf::Quaternion q(quaternion.x, quaternion.y, quaternion.z, quaternion
            .w);
27      tf::Matrix3x3 m(q);
28
29      double yaw_IMU_NED,pitch_IMU_NED,roll_IMU_NED;
30      mavros::UAS::quaternion_to_rpy(quaterniond, roll_IMU_NED,
            pitch_IMU_NED, yaw_IMU_NED);
31
32      //convert quaternion to euler angels
33      m.getEulerYPR(yaw_IMU_NED, pitch_IMU_NED, roll_IMU_NED);
34
35      //Converts NED frame to ENU  frame
36      double  yaw_IMU_ENU =  - yaw_IMU_NED;   // ENU [-pi pi]
37
38      /*
39      3) Get Navigation controller (NC) outputs
40      NC_ref_yaw_ENU    (ENU in [-pi, pi] +ccw -cw)
41      NC_ref_pitch_ENU  (ENU in [-pi, pi] +forward,-backward)
42      NC_ref_roll_ENU   (ENU in [-pi, pi] +rigthward,-leftward)
43      NC_ref_thrust_ENU (ENU in [0 1] +upward,-downward)
44      */
45
46      /*
47      4) Calculate Pixhawk Autopilot commands (PI)
48      PI_ref_yaw_NED    (NED in [-pi, pi] +cw -ccw)
49      PI_ref_pitch_NED  (NED in [-pi, pi] +backward,-forward)
50      PI_ref_roll_NED   (NED in [-pi, pi] +rigthward,-leftward)
51      PI_ref_thrust_ENU (ENU in [0 1] +upward,-downward)
52      */
53
54      double PI_ref_yaw_NED = -(yaw_IMU_ENU + NC_ref_dyaw_ENU);
55      double PI_ref_roll_NED =  NC_ref_roll_ENU;
56      double PI_ref_pitch_NED = -NC_ref_pitch_ENU;
57      double PI_ref_thrust_ENU = NC_ref_thrust_ENU;
58
59      /*
60      5) Send to the Pixhawk through mavros
61      */
62
63      Eigen::Quaterniond quaterniond_NED =  mavros::UAS::
            quaternion_from_rpy(PI_ref_roll_NED, PI_ref_pitch_NED,
            PI_ref_yaw_NED);
64
```

```
65      Eigen::Quaterniond quaterniond_ENU = mavros::UAS::
            transform_orientation_ned_enu(mavros::UAS::
            transform_orientation_baselink_aircraft(quaterniond_NED));
66
67      geometry_msgs::Quaternion orientation_ENU;
68      tf::quaternionEigenToMsg(quaterniond_ENU, orientation_ENU);
69
70      mavros_msgs::AttitudeTarget    attitude_msg;
71      attitude_msg.orientation = orientation_ENU;
72      attitude_msg.thrust = PI_ref_thrust_ENU;
73
74      /*************************************************************/
75      // Publish message into "mavros/setpoint_raw/attitude" topic
76      /*************************************************************/
```

Appendix C

Vision Based Planner Appendix

C-1 Distance estimation

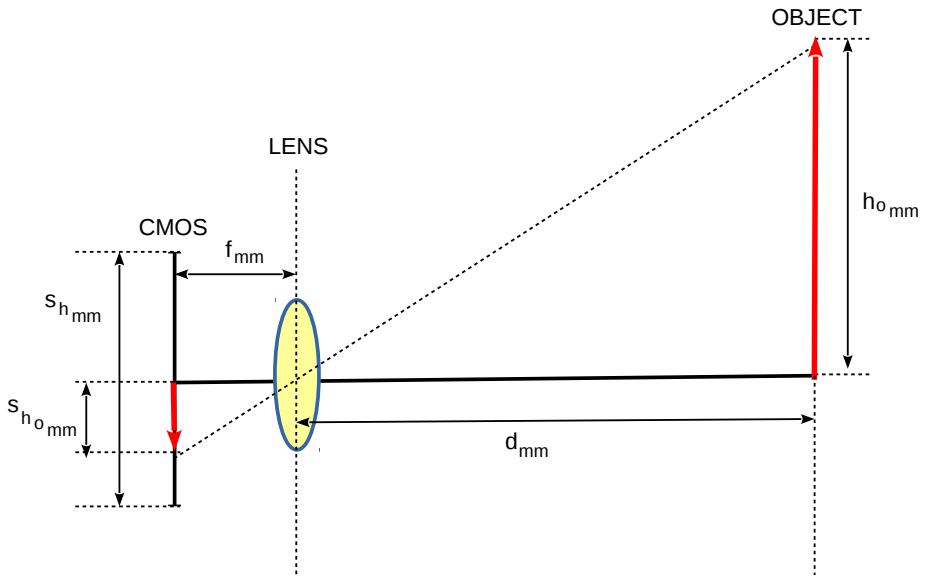


Figure C-1: CMOS, lens and object illustration required to derive the distance between lens and object (d_{mm})

Given Figure C-1 is possible to derive an estimation of the distance between object and camera lens (d_{mm}). To do this two different consideration are done.

- Firstly is possible to find a relation between the real object height ($h_{o_{mm}}$), the corresponding height of the object on the CMOS camera sensor ($s_{h_{o_{mm}}}$) , the absolute focal length and the distance between object and camera lens where all the parameters are expressed in millimeters. The mathematical relation is given by

$$\frac{h_{o_{mm}}}{s_{h_{o_{mm}}}} = \frac{d_{mm}}{f_{mm}} \tag{C-1}$$

  which means that ratio between the real object height and the height of object appearing on the camera sensor is equal to the ratio between the distance between object and camera lens and the absolute focal length.

- Secondly is possible to find a relation between the height of the object on the camera sensor expressed in millimeters and the height of the object in pixel. The need of this relation is due to the fact that it is easier to know the height of an object in pixel rather than the height of an object on the camera sensor in millimeters. The mathematical relation is

$$\frac{s_{h_{o_{mm}}}}{h_{o_{pix}}} = \frac{s_{h_{mm}}}{im_{h_{pix}}} \rightarrow s_{h_{o_{mm}}} = \frac{s_{h_{mm}} h_{o_{pix}}}{im_{h_{pix}}} \tag{C-2}$$

  where ($h_{o_{pix}}$) and ($im_{h_{pix}}$) represent respectively the object height in pixel and the height of the image (number of rows) expressed in pixel.

Combining Eq. (C-1) with Eq. (C-2) is possible to derive an equation able to estimate the distance between object and camera lens as follow

$$d_{mm} = \frac{f_{mm} h_{o_{mm}} im_{h_{pix}}}{s_{h_{mm}} h_{o_{pix}}} \tag{C-3}$$

where the absolute focal length (f_{mm}), the sensor height ($s_{h_{mm}}$) and he image height ($im_{h_{pix}}$) in pixel are available looking at the chosen camera documentation whereas the height of the object is given by the detector. Given equation Eq. (C-3) knowing the desired distance between camera and object ($d^{\star}_{mm}$), it is possible to compute the object height in pixel as follow

$$h_{o_{pix}} = \frac{f_{mm} h_{o_{mm}} im_{h_{pix}}}{s_{h_{mm}} d^{\star}_{mm}} \tag{C-4}$$

C-2 Time derivative of a 2D image point

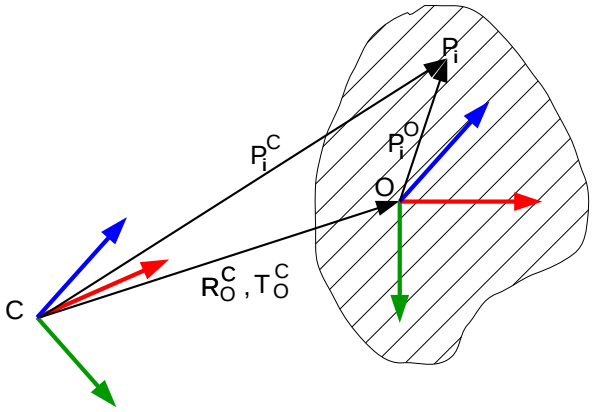


Figure C-2: Camera ($\mathbf{C}$) and object frame ($\mathbf{O}$) representation. The point P_i^O is rigidly attached to the object frame. The latter is thought as a rigid body.

Given a point $P_i^C = \begin{bmatrix} X_i^C & Y_i^C & Z_i^C \end{bmatrix}^T$ representing the coordinate of a point i expressed in the camera frame (C) and a point $P_i^O = \begin{bmatrix} X_i^O & Y_i^O & Z_i^O \end{bmatrix}^T$ attached to the rigid target object frame (O), the equation describing P_i^C as a function of P_i^O is

$$P_i^C = R_O^C P_i^O + T_O^C \tag{C-5}$$

The derivative of this point ($\dot{P_i^C}$) is

$$\dot{P_i^C} = \dot{R}_O^C P_i^O + \cancelto{0}{R_O^C \dot{P_i^O}} + \dot{T}_O^C \tag{C-6}$$

Making the following considerations

- $\cancelto{0}{R_O^C \dot{P_i^O}}$ because the point P_i is rigidly attached to the object frame (O). This condition makes the derivative of point P_i expressed in the object frame equal to zero. If an observer is sitting on the origin of the object frame it will see the point P_i static in time.

- $w_O^{C,C}$ is the angular velocity of the object frame with respect to the camera frame expressed in camera frame coordinate. The latter can be rewritten as

$$\left[\mathbf{w}_O^{C,C}\right]_\times = \dot{R}_O^C R_C^O \rightarrow \dot{R}_O^C = \left[\mathbf{w}_O^{C,C}\right]_\times R_O^C \tag{C-7}$$

where $\left[\mathbf{w}_O^{C,C}\right]_\times$ is the skew symmetric matrix of the vector $w_O^{C,C} \in \mathbb{R}^{3\times 1}$.

$$\left[\mathbf{w}_O^{C,C}\right]_\times = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}_\times = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix} \tag{C-8}$$

- $\dot{T}_O^C$ represents the translational velocity of the object frame with respect to the camera frame in camera frame coordinate ($\mathbf{v}_O^{C,C}$)

is possible to write

$$\begin{aligned} \dot{P_i^C} &= \left[\mathbf{w}_O^{C,C}\right]_\times R_O^C P_i^O + \mathbf{v}_O^{C,C} = \left[\mathbf{w}_O^{C,C}\right]_\times P_i^C + v_O^{C,C} = \\ &= -\left[P_i^C\right]_\times \mathbf{w}_O^{C,C} + \mathbf{v}_O^{C,C} \end{aligned} \tag{C-9}$$

which is equivalent to

$$\dot{P_i^C} = \begin{bmatrix} \dot{X}_i^C \\ \dot{Y}_i^C \\ \dot{Z}_i^C \end{bmatrix} = \begin{bmatrix} I_{3\times 3} & -\left[P_i^C\right]_\times \end{bmatrix} \begin{bmatrix} \mathbf{v}_O^{C,C} \\ \mathbf{w}_O^{C,C} \end{bmatrix} \quad with \quad \left[P_i^C\right]_\times = \begin{bmatrix} 0 & -Z_i^C & Y_i^C \\ Z_i^C & 0 & -X_i^C \\ -Y_i^C & X_i^C & 0 \end{bmatrix} \tag{C-10}$$

where ($\mathbf{v}_O^{C,C}$) and ($\mathbf{w}_O^{C,C}$) represent respectively the translational velocity of the object frame with respect to the camera frame expressed in camera frame coordinate and the rotational velocity of the object frame with respect to the camera frame expressed in camera frame coordinate. Dealing with visual servoing is more interested to know the velocities of the camera frame with respect to the object frame in camera frame coordinate ($\mathbf{v}_C^{C,O}, \mathbf{w}_C^{C,O}$) . The relation between ($\mathbf{v}_C^{C,O}, \mathbf{w}_C^{C,O}$) and ($\mathbf{v}_O^{C,C}, \mathbf{w}_O^{C,C}$) is

$$\begin{aligned} \mathbf{v}_C^{C,O} &= -\mathbf{v}_O^{C,C} \\ \mathbf{w}_C^{C,O} &= -\mathbf{w}_O^{C,C} \end{aligned} \tag{C-11}$$

Thus (C-10) can be rewritten as

$$\dot{P_i^C} = \begin{bmatrix} \dot{X}_i^C \\ \dot{Y}_i^C \\ \dot{Z}_i^C \end{bmatrix} = \begin{bmatrix} -I_{3\times 3} & \left[P_i^C\right]_\times \end{bmatrix} \begin{bmatrix} \mathbf{v}_C^{C,O} \\ \mathbf{w}_C^{C,O} \end{bmatrix} \tag{C-12}$$

which is equivalent to

$$\begin{aligned} \dot{X}_i^C &= -v_{x_C}^{C,O} - w_{y_C}^{C,O} Z_i^C + w_{z_C}^{C,O} Y_i^C \\ \dot{Y}_i^C &= -v_{y_C}^{C,O} - w_{z_C}^{C,O} X_i^C + w_{x_C}^{C,O} Z_i^C \\ \dot{Z}_i^C &= -v_{z_C}^{C,O} - w_{x_C}^{C,O} Y_i^C + w_{y_C}^{C,O} X_i^C \end{aligned} \tag{C-13}$$

Knowing that the $2D$ image coordinates of a generic i point, according to perspective projection, are defined as

$$\begin{aligned} x_{2D_i} &= \frac{X_i^C}{Z_i^C} = \frac{u_i - c_x}{f_x} \\ y_{2D_i} &= \frac{Y_i^C}{Z_i^C} = \frac{v_i - c_y}{f_y} \end{aligned} \tag{C-14}$$

with $u_i = column_i$ and $v_i = row_i$ pixel coordinates. The derivative of the $2D$ image coordinates are

$$\begin{aligned} \dot{x}_{2D_i} &= \frac{\dot{X}_i^C}{Z_i^C} - \frac{X_i^C \dot{Z}_i^C}{{Z_i^C}^2} = \frac{(\dot{X}_i^C - x_{2D_i}\dot{Z}_i^C)}{Z_i^C} \\ \dot{y}_{2D_i} &= \frac{\dot{Y}_i^C}{Z_i^C} - \frac{Y_i^C \dot{Z}_i^C}{{Z_i^C}^2} = \frac{(\dot{Y}_i^C - y_{2D_i}\dot{Z}_i^C)}{Z_i^C} \end{aligned} \tag{C-15}$$

Replacing (C-13) into (C-15) results

$$\begin{aligned} \dot{x}_{2D_i} &= -\frac{v_{x_C}^{C,O}}{Z_i^C} + \frac{x_{2D_i} v_{z_C}^{C,O}}{Z_i^C} + x_{2D_i} y_{2D_i} w_{x_C}^{C,O} - (1 + x_{2D_i}^2) w_{y_C}^{C,O} + y_{2D_i} w_{z_C}^{C,O} \\ \dot{y}_{2D_i} &= -\frac{v_{y_C}^{C,O}}{Z_i^C} + \frac{y_{2D_i} v_{z_C}^{C,O}}{Z_i^C} - x_{2D_i} y_{2D_i} w_{y_C}^{C,O} + (1 + y_{2D_i}^2) w_{x_C}^{C,O} - x_{2D_i} w_{z_C}^{C,O} \end{aligned} \tag{C-16}$$

Rewriting (C-16) in a matrix form results

$$\begin{bmatrix} \dot{x}_{2D_i} \\ \dot{y}_{2D_i} \end{bmatrix} = L_{x_i} \mathbf{V}_C^{C,O} \quad with \quad \mathbf{V}_C^{C,O} = \begin{bmatrix} \mathbf{v}_C^{C,O} \\ \mathbf{w}_C^{C,O} \end{bmatrix} = \begin{bmatrix} v_{x_C}^{C,O} \\ v_{y_C}^{C,O} \\ v_{z_C}^{C,O} \\ w_{x_C}^{C,O} \\ w_{y_C}^{C,O} \\ w_{z_C}^{C,O} \end{bmatrix} \tag{C-17}$$

where L_{x_i} is known as interaction matrix and it is equal to

$$L_{x_i} = \begin{bmatrix} -\frac{1}{Z_i^C} & 0 & \frac{x_{2D_i}}{Z_i^C} & x_{2D_i} y_{2D_i} & -(1 + x_{2D_i}^2) & y_{2D_i} \\ 0 & -\frac{1}{Z_i^C} & \frac{y_{2D_i}}{Z_i^C} & 1 + y_{2D_i}^2 & -x_{2D_i} y_{2D_i} & -x_{2D_i} \end{bmatrix} \tag{C-18}$$

Considering only translational motion Eq. (C-17) becomes

$$\begin{bmatrix} \dot{x}_{2D_i} \\ \dot{y}_{2D_i} \end{bmatrix} = \begin{bmatrix} -\frac{1}{Z_i^C} & 0 & \frac{x_{2D_i}}{Z_i^C} \\ 0 & -\frac{1}{Z_i^C} & \frac{y_{2D_i}}{Z_i^C} \end{bmatrix} \begin{bmatrix} v_{x_C}^{C,O} \\ v_{y_C}^{C,O} \\ v_{z_C}^{C,O} \end{bmatrix} = L_{x_i} \mathbf{v}_C^{C,O} \tag{C-19}$$

where $\mathbf{v}_C^{C,O}$ are the instantaneous translational camera velocities describing the tranlsational velocities of the current camera frame (C) with respect to the object frame (target visual marker O) expressed in the current camera frame (C). L_{x_i} is the interaction matrix relating the derivative of the $2D$ image point ($\dot{m}_{2D_i}^{\tau}$) with the instantaneous translational camera velocities ($\mathbf{v}_C^{C,O}$).

C-3 Derivation of the distance between two $2D$ image coordinate points

Given a generic distance among two points k and j along x_{2D} and y_{2D} direction

$$\begin{aligned} d_{x_{2D_{k,j}}} &= x_{2D_k} - x_{2D_j} \\ d_{y_{2D_{k,j}}} &= y_{2D_k} - y_{2D_j} \end{aligned} \tag{C-20}$$

the derivative in time is

$$\begin{aligned} \dot{d}_{x_{2D_{k,j}}} &= \dot{x}_{2D_k} - \dot{x}_{2D_j} \\ \dot{d}_{y_{2D_{k,j}}} &= \dot{y}_{2D_k} - \dot{y}_{2D_j} \end{aligned} \tag{C-21}$$

Knowing from Appendix C-2 that the derivative of a generic i $2D$ image feature is given by

$$\begin{bmatrix} \dot{x}_{2D_i} \\ \dot{y}_{2D_i} \end{bmatrix} = \begin{bmatrix} -\frac{1}{Z_i^C} & 0 & \frac{x_{2D_i}}{Z_i^C} & x_{2D_i} y_{2D_i} & -(1+x_{2D_i}^2) & y_{2D_i} \\ 0 & -\frac{1}{Z_i^C} & \frac{y_{2D_i}}{Z_i^C} & 1+y_{2D_i}^2 & -x_{2D_i} y_{2D_i} & -x_{2D_i} \end{bmatrix} \begin{bmatrix} \mathbf{v_C^{C,O}} \\ \mathbf{w_C^{C,O}} \end{bmatrix} \tag{C-22}$$

combining Eq. (C-21) with Eq. (C-22) is possible to rewrite the derivative of the distance as

$$\begin{aligned} \dot{d}_{x_{2D_{k,j}}} &= \Big(-\frac{1}{Z_k^C} + \frac{1}{Z_j^C} \Big) v_{x_C}^{C,O} + \Big(\frac{x_{2D_k}}{Z_k^C} - \frac{x_{2D_j}}{Z_j^C} \Big) v_{z_C}^{C,O} + \Big(x_{2D_k} y_{2D_k} - x_{2D_j} y_{2D_j} \Big) w_{x_C}^{C,O} + \\ &+ \Big(-(1+x_{2D_k}^2) + (1+x_{2D_j}^2) \Big) w_{y_C}^{C,O} + \Big(y_{2D_k} - y_{2D_j} \Big) w_{z_C}^{C,O} \\ \dot{d}_{y_{2D_{k,j}}} &= \Big(-\frac{1}{Z_k^C} + \frac{1}{Z_j^C} \Big) v_{y_C}^{C,O} + \Big(\frac{y_{2D_k}}{Z_k^C} - \frac{y_{2D_j}}{Z_j^C} \Big) v_{z_C}^{C,O} + \Big((1+y_{2D_k}^2) - (1+y_{2D_j}^2) \Big) w_{x_C}^{C,O} + \\ &+ \Big(-x_{2D_k} y_{2D_k} + x_{2D_j} y_{2D_j} \Big) w_{y_C}^{C,O} + \Big(-x_{2D_k} + x_{2D_j} \Big) w_{z_C}^{C,O} \end{aligned} \tag{C-23}$$

Considering only translational motion Equation (C-23) becomes

$$\begin{aligned} \dot{d}_{x_{2D_{k,j}}} &= \Big[(-\tfrac{1}{Z_k^C} + \tfrac{1}{Z_j^C}) \quad 0 \quad (\tfrac{x_{2D_k}}{Z_k^C} - \tfrac{x_{2D_j}}{Z_j^C}) \Big] \mathbf{v}_C^{C,O} \\ \dot{d}_{y_{2D_{k,j}}} &= \Big[0 \quad (-\tfrac{1}{Z_k^C} + \tfrac{1}{Z_j^C}) \quad (\tfrac{y_{2D_k}}{Z_k^C} - \tfrac{y_{2D_j}}{Z_j^C}) \Big] \mathbf{v}_C^{C,O} \end{aligned} \tag{C-24}$$

Bibliography

[1] J. L. Sánchez López, *A General Architecture for Autonomous Navigation of Unmanned Aerial Systems*. PhD thesis, Industriales, 2017.

[2] J. L. Sanchez-Lopez, R. A. S. Fernández, H. Bavle, C. Sampedro, M. Molina, J. Pestana, and P. Campoy, "Aerostack: An architecture and open-source software framework for aerial robotics," in *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pp. 332–341, IEEE, 2016.

[3] J. Pestana, I. Mellado-Bataller, C. Fu, J. L. Sanchez-Lopez, I. F. Mondragon, and P. Campoy, "A general purpose configurable navigation controller for micro aerial multi-rotor vehicles," in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pp. 557–564, IEEE, 2013.

[4] G. Szafranski and R. Czyba, "Different approaches of pid control uav type quadrotor," 2011.

[5] R. Munoz-Salinas, "Aruco: a minimal library for augmented reality applications based on opencv," *Universidad de Crdoba*, 2012.

[6] J.-Y. Bouguet, "Matlab camera calibration toolbox." http://www.vision.caltech.edu/bouguetj/calib_doc/, 2000.

[7] G. Bradski and A. Kaehler, "Opencv camera calibration." http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html.

[8] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[9] M. Verhaegen and V. Verdult, *Filtering and system identification: a least squares approach*. Cambridge university press, 2007.

[10] F. Chaumette and S. Hutchinson, "Visual servo control. ii. advanced approaches [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 109–118, 2007.

[11] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches," *IEEE Robotics & Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[12] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE transactions on robotics and automation*, vol. 12, no. 5, pp. 651–670, 1996.

[13] D. Lee, T. Ryan, and H. J. Kim, "Autonomous landing of a vtol uav on a moving platform using image-based visual servoing," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 971–976, IEEE, 2012.

[14] P. Serra, R. Cunha, T. Hamel, D. Cabecinhas, and C. Silvestre, "Landing of a quadrotor on a moving target using dynamic image-based visual servo control," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1524–1535, 2016.

[15] C. V. Martínez Luna, *Visual Tracking, Pose Estimation, and Control for Aerial Vehicles*. PhD thesis, Industriales, 2013.

[16] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *ieee Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 313–326, 1992.

[17] P.-J. Bristeau, F. Callou, D. Vissiere, and N. Petit, "The navigation and control technology inside the ar. drone micro uav," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 1477–1484, 2011.

[18] "Px4 software-in-the-loop." http://github.com/PX4/sitl_gazebo.

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, 2009.

[20] A. Martinez and E. Fernández, *Learning ROS for robotics programming*. Packt Publishing Ltd, 2013.

[21] M. Molina, "An execution engine for aerial robot mission plans," technical report, ETSI_Informatica, June 2017.

[22] G. de Fermín Cordeiro, "Mission plan interpretation for the aerostack software framework." June 2017.

[23] A. C. Portela, "Development of a behavior management system for the aerial robot software framework aerostack." June 2017.

[24] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, 2012.

Glossary

List of Acronyms

UAV	Unmanned Aerial Vehicle
GPS	Global Positioning System
EKF	Extended Kalman Filter
MOCAP	Motion Capture System
IMU	Inertial Measurement Unit
ROS	Robot Operating System
ENU	East North Up
NED	North East Down
PID	Proportional Integral Derivative controller
PD	Proportional Derivative controller
2DOF	Two degree of freedom
IBVS	Image Based Visual Servoing