



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Alessio Zanchettin

# Autonomous Quadrotor Landing on a Moving Platform with only Onboard Sensing and Computing

Master Thesis

Robotics and Perception Group  
University of Zurich

## Supervision

Davide Falanga  
Junje Zhang  
Prof. Dr. Davide Scaramuzza

September 2016



# Contents

<b>Abstract</b>	<b>iii</b>
<b>Nomenclature</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Related Work . . . . .	2
1.1.1 Landing on a static platform . . . . .	2
1.1.2 Landing on a moving platform . . . . .	2
1.1.3 MPC for quadrotors . . . . .	4
1.2 MBZIRC challenge . . . . .	5
1.2.1 The Arena . . . . .	5
1.2.2 Landing platform . . . . .	6
<b>2 General Overview</b>	<b>7</b>
2.1 SVO & MSF . . . . .	8
2.1.1 SVO . . . . .	8
2.1.2 MSF . . . . .	9
2.2 Control . . . . .	9
2.2.1 High level control . . . . .	9
2.2.2 Low level control . . . . .	10
2.3 Base detection & tracking . . . . .	11
2.4 State machine . . . . .	12
2.5 Trajectory generation . . . . .	12
<b>3 Base detection and tracking</b>	<b>13</b>
3.1 Prediction update: non-holonomic model . . . . .	15
3.1.1 Straight and circular path . . . . .	17
3.1.2 Infinity shape path . . . . .	17
3.2 Measurement update . . . . .	19
3.2.1 Platform pose estimation at high altitudes . . . . .	20
3.2.2 Platform pose estimation at low altitudes . . . . .	26
3.2.3 Covariance Estimation . . . . .	28
<b>4 State machine</b>	<b>31</b>
4.1 Takeoff and searching for the base . . . . .	31
4.2 Following the base . . . . .	33
4.2.1 Understand type of movement . . . . .	33
4.2.2 Calculate future positions of the moving platform . . . . .	39

4.2.3	Select moment to land . . . . .	41
4.3	Approaching the base . . . . .	45
4.4	Align with the base . . . . .	47
4.5	Landing on the base . . . . .	48
<b>5</b>	<b>Trajectory generator</b>	<b>51</b>
5.1	Rapid Trajectory . . . . .	53
5.1.1	Dynamic model . . . . .	53
5.1.2	Optimal control problem . . . . .	54
5.1.3	Cost function . . . . .	55
5.1.4	Constraints . . . . .	56
5.1.5	Feasibility check . . . . .	56
5.1.6	Compute the acceleration . . . . .	57
5.2	Minimum snap trajectory . . . . .	58
5.3	Problems with the trajectory generation . . . . .	59
5.3.1	Last chance solution . . . . .	59
5.3.2	Too short final time . . . . .	60
<b>6</b>	<b>Experiments</b>	<b>61</b>
6.1	Real world hardware . . . . .	61
6.1.1	Quadrotor . . . . .	61
6.1.2	Moving platform . . . . .	62
6.2	Simulation . . . . .	62
6.3	SVO . . . . .	63
6.3.1	Front looking vs down looking . . . . .	64
6.4	Base detection and tracking . . . . .	67
6.4.1	From high altitude . . . . .	67
6.4.2	From low altitude . . . . .	69
6.5	Trajectory generation . . . . .	70
6.5.1	Acceleration estimation . . . . .	70
6.6	Landing on a moving platform . . . . .	73
<b>7</b>	<b>Discussion</b>	<b>74</b>
7.1	Conclusion . . . . .	74
7.2	Future Work . . . . .	75
7.2.1	SVO with multiple cameras . . . . .	75
7.2.2	State estimation fuse multiple sensors . . . . .	75
7.2.3	Change the controller . . . . .	75
7.2.4	Cross detector . . . . .	76

# Abstract

This thesis focused on autonomous quadrotor landing on a moving platform. The aerial robot employs a forward-looking camera to perform state estimation, and a down looking-camera to detect and observe the landing platform carried by a mobile robot moving independently inside an arena. Measurements from the downlooking camera are combined with a proper dynamic model in order to estimate position and velocity of the moving platform.

The overall goal is to design a complete framework to perform the entire task: area exploration to look for the base; finding and approaching the platform, while keeping it within the camera's field of view; finally landing on it, minimizing the error in position and velocity.

The frameworks consists of several modules that perform different functions and collaborate together to complete the mission. All the computation run onboard, and so the quadrotor can perform this task fully autonomously.

The system is validated in the real world experiments: the vehicle successfully landed on the moving platform during outdoor flight tests.



# Nomenclature

## Notation

<b>J</b>	Jacobian
<i>r</i>	position of the frame $B$ with respect to frame $W$
$\mathbf{T}_{WB}$	coordinate transformation from frame $B$ to frame $W$
$\mathbf{R}_{WB}$	orientation of $B$ with respect to $W$
$\hat{\mathbf{w}}_{WB}$	skew symmetric matrix
$\mathbf{c}$	thrust vector with respect to frame $B$
$\mathbf{g}$	gravity with respect to frame $W$

Scalars are written in lower case letters (*a*), vectors in lower case bold letters (**a**) and matrices in upper case bold letters (**A**).

## Acronyms and Abbreviations

RPG	Robotics and Perception Group
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
MAV	Micro Aerial Vehicle
ROS	Robot Operating System
DoF	Degree of Freedom
IMU	Inertial Measurement Unit
EKF	Extended Kalman Filter
SVO	Semidirect Visual Odometry
MSF	Multi Sensor Fusion
MBZIRC	Mohamed Bin Zayed International Robotics Challenge

# Chapter 1

## Introduction

Unmanned Aerial Vehicles (UAVs) are, nowadays, accessible to all kind of users, thanks to their agility and versatility, they can be used in a wide range of applications. For many of these it is necessary an autonomous landing of the UAV on a platform using only onboard sensors. As a matter of fact, one of the major drawback of current civil Micro Air Vehicles (MAVs) is the limited flight time: automated landing systems (along with suitable recharging platforms) enable longer UAV missions with greater autonomy.

Furthermore these applications often require the landing target to be moving: for example, it can be a car during a reconnaissance. Therefore the MAV must be able to perform a precise landing maneuver over a specific moving platform.

Highly accurate localization is required in order to allow the MAV to land precisely over the platform. Most of UAVs are equipped with a GPS which only offers a precision up to 5 meters, and landing with such an uncertain state estimation can easily lead to failure. On the other hand, many applications allow the usage of other sensors, such as onboard cameras: vision based approaches, to estimate the state both of the UAV and of the moving base, are promising in this respect.

In this work we present a complete framework to let a quadrotor autonomously and safely land on a moving platform using only onboard sensing and computing. The main parts of the framework are:

- self localization and state estimation of the UAV;
- detection, tracking and state estimation of the moving landing target;
- dynamic trajectory planning to perform a precise and smooth land on the target.

## 1.1 Related Work

During the last decades several methods were developed in order to achieve automatic landing for UAVs. Usually, in these projects calculations are done by ground stations, which allows great processing power, but lead to restrictions in autonomy on the UAV.

### 1.1.1 Landing on a static platform

At the beginning the research is focused on landing on a static platform. Hardware and techniques used to achieve the successful completion of the task were various. In [1], Saripalli et al. present a vision-based autonomous landing algorithm using big vehicles that can carry industrial sensors and high performance processors. This work uses hardware very far from the one we want to use, but it one of the first approaches to find a solution to this problem.

Other works, i.e. [2] and [3], use little UAVs with cameras, estimating the pose of the quadrotor only with respect to the landing base. This consists of a single tag, therefore these frameworks are not robust to the loss of the tag and have a very noisy state estimation for the quadrotor.

Another similar work is done by Herisse in [4], where the optical flow is used for hovering flight and vertical landing control.

The main difference with respect to our approach is that these frameworks only tackle the landing maneuver, and the final target is always in the field of view (f.o.v) of the camera. This assumption does not hold in our case.

Other papers present theoretical algorithm to perform a smooth and precise landing, but are only tested in simulation. In [5] the authors present a landing framework based on N-points algorithm and orthogonalization to estimate the state of the aircraft. In [6] a theoretical optical guided landing control system and its corresponding guidance control law are presented.

Mellinger in [7] tackles a similar problem: landing on tilted surface on which the quadrotor must perch. He uses a motion capture system in order to have both UAV and target state estimate. His algorithm consists of a precomputed trajectory followed by a position-attitude control based on the linearized model of the quadrotor.

An interesting part of this work is the subdivision of the task in smaller parts in which trajectories and control are different in order to increase the robustness of the whole maneuver.

### 1.1.2 Landing on a moving platform

More interesting for the purpose of this thesis, are researches about landing on a moving platform.

Wenzel in [8] is performing tracking and landing on a moving base with a small quadrotor. All the experiments are indoors, because of the use of IR cameras, which are not robust to outdoor conditions, due to direct sunlight. Precise and consistent results are achieved with a platform moving both in a circular path or emulating a ship turning on water, with velocities up to  $0.4 \text{ m s}^{-1}$ .

In [9] Lee et al. are using visual servoing to perform the landing maneuver. A feedback control law based on the position of the target in the camera image is derived and the method is evaluated with a landing platform moving in a straight line with velocities of up to  $0.07 \text{ m s}^{-1}$ . To control the quadrotor to the landing site Sliding Mode Control is used. This method can deal with non linearity of the dynamics and external modeled noise (like the model of the ground effect force).

In [10] the authors propose a method to let a quadrotor land on a moving platform which can be easily found in the image of the onboard camera using color segmentation and blob detection. More specifically the platform has a color that no other object in the environment has. Also an omnidirectional camera is used to search for the platform in the surrounding environment. Given the measurement of the position of the camera he implements an Extended Kalman Filter (EKF) to reduce the noise and predict the future position of the target. The formulation of the EKF is oversimplified, because the platform is allowed to move only in a straight line. Once the future position is estimate, a trajectory, namely position and velocity, is computed from the initial pose of the quadrotor to the final intersection point. A velocity-attitude control is implemented to follow the trajectory without replanning.

Vlantis et. al in [11] study the problem of landing a quadrotor on an inclined moving platform. The UAV carries a forward looking camera to detect and observe the landing platform. In order to complete the task, a discrete-time non-linear Model Predictive Controller (MPC) [12] is developed. The control algorithm optimizes both the trajectories and the time horizon, while respecting input and state constraints.

The cost function of the MPC consists of different therms weighted with dynamic coefficients (function of the relative position between UAV and moving platform). More specifically, these therms are the execution time, the state of the quadrotor (position, orientation, velocity, body-rates), the smoothness and aggressiveness of the control inputs, and other factors regarding the landing task, such as the alignment between the states of UAV and moving platform (relative position, orientation, velocity) and the fact that the center of the platform should be kept within the camera's field of view during the approaching phase.

The major drawback of this approach is that the MPC is computationally very expensive and it is not possible to run the algorithm on-board: a ground station that carries the huge amount of calculation that MPC requires is necessary.

The main conclusions from the analysis of these related works is that to design a landing framework we need:

- a good state estimation of both the UAV and the moving platform;
- a subdivision of the whole task in subtasks, and a manager that decides

in each moment which of these stages must be completed;

- a MPC algorithm that control the quadrotor to complete the assigned task. The algorithm should increase the robustness updating continuously the future actions that must be applied to the UAV.

### 1.1.3 MPC for quadrotors

Several papers have been written on MPC applied to control of the quadrotor.

In [13] Neunert et.al. present a framework for real-time, unconstrained, nonlinear MPC. The algorithm combines trajectory optimization and tracking control in a unified approach. It solves the MPC problem using repeatedly a method called Sequential Linear Quadratic, generating feedforward and feedback controls actions. This method allows agile flight maneuvers with accurate tracking. All the calculations are made on an on-board Intel i7 CPU achieving a rate slightly below 40 Hz. No details about the performance of such a controller with lower computational power is provided.

In the paper [14] Bangura presents a solution to on-board trajectory tracking control for quadrotors. The proposed approach combines a standard control paradigm for attitude and a high-level trajectory tracking with a MPC strategy. In order to reduce the complexity, the system is feedback linearized obtaining an equivalent linear model to which the MPC framework can be applied. Also in this case only the computation related to MPC are done onboard.

Mueller in [15] and later in [16] presents a method for rapid generation and feasibility verification of trajectories for quadrotors. The motion primitives are defined by the quadrotor's initial state at time  $t_0$  (position, velocity, acceleration), the desired motion duration  $T$ , and any combination of components of the quadrotor's position, velocity and acceleration at time  $t_0 + T$ . The trajectory are the solution of an optimization problem which minimize a cost function related to input aggressiveness, and checks if it is feasible both with respect to input and state constraints. Millions of motion primitives can be generated and evaluated per second, and so the best feasible one can be picked and followed.

Table 1.1 summarizes the work done in previous research on this topic.

	<b>Lee 2012</b>	<b>Wenzel 2011</b>	<b>Kim 2016</b>	<b>Vlantis 2015</b>
<b>velocity platform [m/s]</b>	<b>0.07</b>	<b>0.4</b>	<b>0.5</b>	<b>0.5</b>
<b>outdoor testing</b>	<b>X</b>	<b>X</b>	<b>✓</b>	<b>✓</b>
<b>quad indip. state estim.</b>	<b>X</b>	<b>X</b>	<b>✓</b>	<b>✓</b>
<b>platform dynamics</b>	<b>X</b>	<b>X</b>	<b>✓</b>	<b>✓</b>
<b>replanning</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>✓</b>
<b>onboard computation</b>	<b>X</b>	<b>✓</b>	<b>X</b>	<b>X</b>

Table 1.1: Summary table of the results achieved in previous works on autonomous landing on a moving platform.

## 1.2 MBZIRC challenge

The main reason for the development of this thesis is the participation in the Mohamed Bin Zayed International Robotics Challenge [17]. MBZIRC is an international robotics competition, held every two years, that provides an ambitious and technologically demanding set of challenges, that aim at inspiring the future of robotics.

The competition is composed of 3 challenges and in this thesis we develop a framework to complete challenge number 1 which consists in an UAV landing on a moving ground vehicle. The specifications of the challenge are:

- Max Duration: 20 minutes;
- UAV initial condition: the participating team positions the UAV in stationary mode on the ground at the start location;
- UGV initial condition: the ground vehicle is driven into the arena and placed at a random location along the track.

### 1.2.1 The Arena

The challenge will be performed in an arena with the following characteristics 1.1:

- Outdoor open arena with GPS access.
- Dimension: approximate 100 m × 60 m;
- Track: width 3 m, in the shape of a figure 8 (or infinity shape), with boundary marked with white paint.
- Terrain: relatively smooth;

- UAV initial start location: 10 m away from the arena.

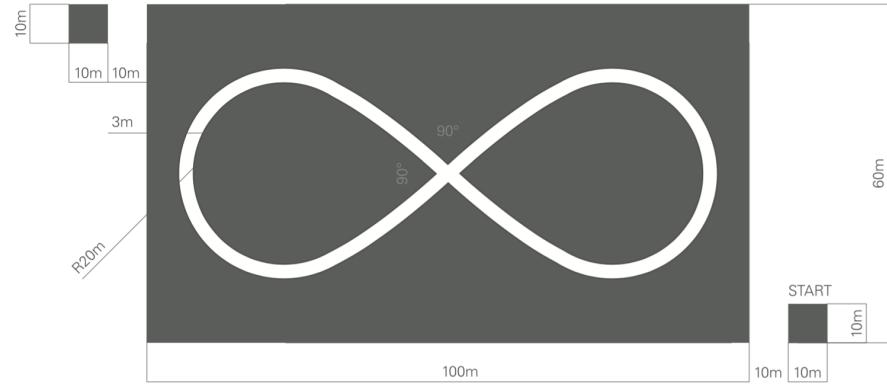


Figure 1.1: Arena of the challenge

### 1.2.2 Landing platform

The landing platform is mounted on a ground vehicle of approximate dimensions  $2.5 \text{ m} \times 1.5 \text{ m} \times 1.5 \text{ m}$ . The moving car starts at a constant speed of  $15 \text{ km h}^{-1}$ , then it reduces the speed to  $6 \text{ km h}^{-1}$  after 6 minutes and to  $5 \text{ km h}^{-1}$  after 12 minutes.

The landing platform is made of a ferrous surface to enable docking using magnetic or suction units. It is a square of dimensions  $1.5 \text{ m} \times 1.5 \text{ m}$ , and approximately 1.5 m above ground, positioned above the vehicle.

The landing zone inside the landing area is a circle of 1 m diameter. The center of the circle is indicated by a cross. The landing area, the landing zone and the cross are shown in 1.2.

A landing is considered successful when the point of contact of the UAV is within the landing circle, with propulsion off and rotors not spinning.

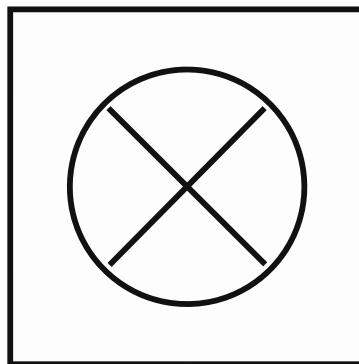


Figure 1.2: Design of the platform in which the quadrotor must land on

## Chapter 2

# General Overview

Our framework consists of several parts, each one dedicated to a different sub-task and communicating with the others in order to achieve the final mission of landing on a moving platform. Figure 2.1 shows the principal components of our framework. In this chapter we provide a brief introduction on each part, and in the following chapters we discuss in details the parts developed in this thesis.

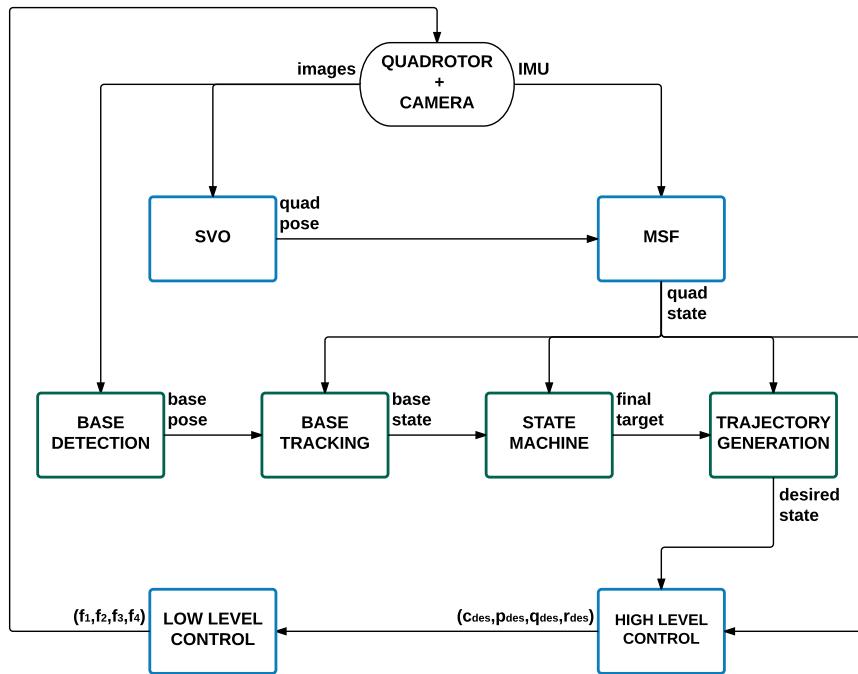


Figure 2.1: Pipeline of the framework. In blue the modules already implemented. In green the modules developed in this work.

Our modular system, utilizes the Robot Operating System (ROS) [18] for interprocess communication. The platforms used in this project are capable of autonomous, vision-based flight utilizing only their on-board hardware resources and they do not need to rely on external communication infrastructure to accomplish their tasks. This is of critical importance in a field robotics scenario with uncertain communication reliability like the MBZIRC, , and our approach enables our platforms to perform their missions autonomously without any dependence on external communication.

## 2.1 SVO & MSF

The backbone of our system is an accurate state estimation pipeline provided by the visual odometry algorithm: Semidirect Monocular Visual Odometry (SVO) [19]. It provides a precise and robust pose estimate (3D position and orientation), and it runs at up to 55 frames per second on the onboard embedded computer of our flying robots. This vision-based pose estimate is fused with inertial data from an Intertial Measurement Unit (IMU) using the Multi-Sensor Fusion framework (MSF) [20].

The fusion of visual and inertial information provides a full state estimate of the UAVs.

### 2.1.1 SVO

SVO implements a semi-direct monocular visual odometry algorithm that estimates the motion of a camera in real time using sequential images. It combines the advantages of features extraction methods and direct approaches.

The standard techniques consist of extracting a sparse set of features in each image, matching them in successive frames using invariant feature descriptors, reconstructing camera motion and structure using epipolar geometry and finally, refining the pose and structure through reprojection error minimization.

On the other hand, appearance-based, or direct, methods estimate structure and motion directly from intensity values of the image: the camera pose relative to the previous frame is found through minimizing the photometric error between pixels.

The semi-direct approach computes an initial guess of the relative camera motion and the feature correspondences using direct methods and concludes with a feature-based nonlinear reprojection-error refinement. This technique increases the computational speed due to the lack of feature-extraction at every frame (this operation is only required when a key-frame is selected to initialize new 3D points). Furthermore, it increases robustness and precision using many small patches instead of few large planar patches. A new 3D point is added to the set used for motion estimation when its depth uncertainty becomes small enough. To estimate it, a probabilistic depth-filter is initialized for each 2D feature for which the corresponding 3D point is to be estimated. The filters are initialized

with a large uncertainty in depth and at every subsequent frame it is updated in a Bayesian fashion.

### 2.1.2 MSF

MSF implements an Iterated Extended Kalman Filter (IEKF) [21] over variable sized windows of updates. In the IEKF the state prediction is driven by IMU data, while the update step can be of any nature. In our case we use the pose estimation given by SVO.

MSF has a modular structure that can support and merge an arbitrary number of sensors providing relative or absolute measurements (pose, position, pressure, etc). Furthermore, it estimates the extrinsic calibration between sensors and tracks the cross covariance terms for relative updates. It also has an outlier rejection module for the update measures.

MSF can compensate for unknown and changing delays implementing further propagation: the state is predicted with IMU data and whenever it receives an update step (usually in the past because of delays) it collocates this measurement in a ring-buffer that considers the moment this data was taken. Then it propagates this measure in time in order to update the current estimation and covariance based on the past data. With this function the framework can give state estimation at IMU rate and without delay.

## 2.2 Control

To control the flight of the quadrotor we need an algorithm that given the current state of the UAV and a final desired state it calculate the input that bring and stabilize the quad on the desired final condition.

The controller of the quadrotor is split into a high-level part and a low-level part: the former enables the tracking of a desired pose and velocity and gives the input to the latter that tracks a desired thrust and body rates.

### 2.2.1 High level control

The high level control takes as input:

- $([\hat{x}, \hat{y}, \hat{z}], [\hat{\phi}, \hat{\theta}, \hat{\psi}], [\hat{v}_x, \hat{v}_y, \hat{v}_z], [\hat{p}, \hat{q}, \hat{r}])$ : the current state estimate (position, orientation, velocity, angular velocity, respectively) , computed in the previous module;
- $([x_{\text{ref}}, y_{\text{ref}}, z_{\text{ref}}], [v_{x,\text{ref}}, v_{y,\text{ref}}, v_{z,\text{ref}}], [a_{x,\text{ref}}, a_{y,\text{ref}}, a_{z,\text{ref}}], [\psi_{\text{ref}}])$ : a reference state (position, velocity, acceleration, yaw) , that can be sampled from a trajectory that the UAV should track to perform a task.

The outputs of this module are:

- $c_{\text{des}}$ : the desired mass-normalized thrust;
- $(p_{\text{des}}, q_{\text{des}}, r_{\text{des}})$ : the desired body rates;

which are sent to the low-level controller.

The high level controller is composed by a position controller followed by an attitude controller, running at 50Hz.

### Position Controller

PD controller with feedback terms on the reference position and velocity and feedforward on the reference acceleration:

$$\begin{bmatrix} a_{x,\text{des}} \\ a_{y,\text{des}} \\ a_{z,\text{des}} \end{bmatrix} = \mathbf{P}_{\text{pos}} \begin{bmatrix} x_{\text{ref}} - \hat{x} \\ y_{\text{ref}} - \hat{y} \\ z_{\text{ref}} - \hat{z} \end{bmatrix} + \mathbf{D}_{\text{pos}} \begin{bmatrix} v_{x,\text{ref}} - \hat{v}_x \\ v_{y,\text{ref}} - \hat{v}_y \\ v_{z,\text{ref}} - \hat{v}_z \end{bmatrix} + \begin{bmatrix} a_{x,\text{ref}} \\ a_{y,\text{ref}} \\ a_{z,\text{ref}} - g \end{bmatrix}, \quad (2.1)$$

where  $\mathbf{P}_{\text{pos}} = \text{diag}(p_{xy}, p_{xy}, p_z)$  and  $\mathbf{D}_{\text{pos}} = \text{diag}(d_{xy}, d_{xy}, d_z)$  are positive-definite gain matrices.

Now is very simple to derive the desired normalized thrust  $c_{\text{des}}$ : it is the projection of  $\mathbf{a}_{\text{des}}$  on the current  $z$  axis of the UAV:

$$c_{\text{des}} = \mathbf{a}_{\text{des}} \cdot \mathbf{e}_z^B \quad (2.2)$$

### Attitude Controller

The output of this controller is  $a_{\text{des}}$  and together with  $\psi_{\text{ref}}$  determine two degrees of freedom of the body orientation  $(\phi_{\text{des}}, \theta_{\text{des}}, \psi_{\text{des}})$ : since the quadrotor can only accelerate along the  $z$  direction of the body, we want to align this axis with the desired acceleration  $\mathbf{a}_{\text{des}}$ , so it enforces both  $\phi_{\text{des}}$  and  $\theta_{\text{des}}$ . The third degree of freedom is given by  $\psi_{\text{ref}}$ .

With some geometric calculations it's easy to define  $(p_{\text{des}}, q_{\text{des}}, r_{\text{des}})$ : these values are function of the current orientation  $(\hat{\phi}, \hat{\theta}, \hat{\psi})$ , the desired final orientation of the  $z$  axis  $\mathbf{e}_{z,\text{des}}^B = \frac{\mathbf{a}_{\text{des}}}{\|\mathbf{a}_{\text{des}}\|}$  and the desired final yaw  $\psi_{\text{des}} = \psi_{\text{res}}$ .

We refer to [22] for further details.

### 2.2.2 Low level control

The low level controller takes as input:

- $c_{\text{des}}$ : the desired normalized thrust
- $(p_{\text{des}}, q_{\text{des}}, r_{\text{des}})$ : the desired body rates
- $(\hat{p}, \hat{q}, \hat{r})$ : the current estimate angular velocity

and gives as output:

- $(f_{1,\text{des}}, f_{2,\text{des}}, f_{3,\text{des}}, f_{4,\text{des}})$ : the desired rotor thrusts.

We can compute the desired torques  $\tau_{\text{des}}$  with the feedback linearized scheme:

$$\begin{bmatrix} \tau_{p,\text{des}} \\ \tau_{q,\text{des}} \\ \tau_{r,\text{des}} \end{bmatrix} = \mathbf{J} \mathbf{P}_{\text{att}} \begin{bmatrix} p_{\text{des}} - \hat{q} \\ q_{\text{des}} - \hat{q} \\ r_{\text{des}} - \hat{r} \end{bmatrix} + \begin{bmatrix} \hat{q} \\ \hat{q} \\ \hat{r} \end{bmatrix} \times \mathbf{J} \begin{bmatrix} \hat{q} \\ \hat{q} \\ \hat{r} \end{bmatrix}, \quad (2.3)$$

where  $\mathbf{P}_{\text{att}} = \text{diag}(p_{qp}, p_{qp}, p_r)$  is a definite-positive gain matrix and  $\mathbf{J} = \text{diag}(J_{xx}, J_{yy}, J_{zz})$  is the inertia matrix for rotation around the center of mass. Now to find the thrusts for each rotor we have to solve:

$$\begin{bmatrix} f_{1,\text{des}} \\ f_{2,\text{des}} \\ f_{3,\text{des}} \\ f_{4,\text{des}} \end{bmatrix} = \begin{bmatrix} \frac{1}{4\lambda_1} \left( mc_{\text{des}} + \frac{\tau_{r,\text{des}}}{\kappa} - \frac{\sqrt{2}\tau_{q,\text{des}}}{l} + \frac{\sqrt{2}\tau_{p,\text{des}}}{l} \right) \\ \frac{1}{4\lambda_2} \left( mc_{\text{des}} - \frac{\tau_{r,\text{des}}}{\kappa} - \frac{\sqrt{2}\tau_{q,\text{des}}}{l} - \frac{\sqrt{2}\tau_{p,\text{des}}}{l} \right) \\ \frac{1}{4\lambda_3} \left( mc_{\text{des}} + \frac{\tau_{r,\text{des}}}{\kappa} + \frac{\sqrt{2}\tau_{q,\text{des}}}{l} - \frac{\sqrt{2}\tau_{p,\text{des}}}{l} \right) \\ \frac{1}{4\lambda_4} \left( mc_{\text{des}} - \frac{\tau_{r,\text{des}}}{\kappa} + \frac{\sqrt{2}\tau_{q,\text{des}}}{l} + \frac{\sqrt{2}\tau_{p,\text{des}}}{l} \right) \end{bmatrix}, \quad (2.4)$$

where  $\kappa$  is the rotor-torque coefficient and  $\lambda_i$  are the rotor fitness coefficients,  $l$  is the arm length between the center of mass and the point in which the thrust is applied.

Depending on the chosen orientation of the body frame we have a different mapping between torques  $\tau_{i,\text{des}}$  and thrusts  $f_j$ . Figure 2.2 shows how our coordinate system is oriented with respect to the 4 propellers.

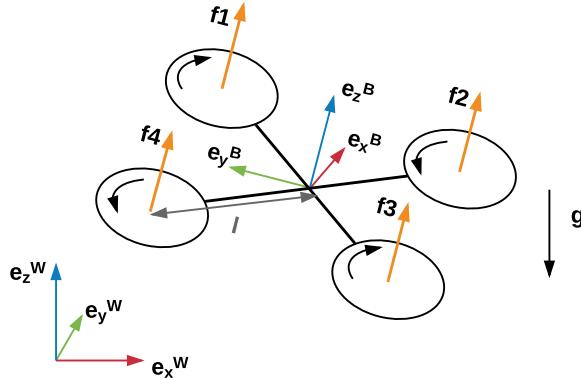


Figure 2.2: Quadrotor with body coordinate frame and thrust forces.

## 2.3 Base detection & tracking

In order to land on the moving platform, it is necessary to know where the base is at a specific time  $t$ . To do so we use images from the onboard camera to localize where the platform is with respect to the quadrotor. Then, given the

position of the UAV with respect to the world frame, we can reconstruct the pose of the moving base in the global coordinate frame.

Now if a mathematical model of how the platform should move in the world frame is available, we can combine the noisy information from the measurements with the theoretical pose that it should have to compute a better estimation of the state of the platform.

Furthermore, we can predict where the platform will be in the future and use this information to plan in advance where the quadrotor must go to complete the task.

This module will be discussed extensively in Chapter 3, with explanation on all the steps we perform to have the final estimation of the base's state.

## 2.4 State machine

A state machine is required to differentiate the behavior of the quadrotor in the various phases of the task. This module implements the manager that decides in which stage the UAV is, based on its pose, the position of the base.

The main output of this module is a final desired target that the quadrotor must reach to complete the current stage and the time  $T$  it should spend to do it.

This module will be discussed extensively in Chapter 4.

## 2.5 Trajectory generation

This module of the framework takes as inputs the quadrotor state estimation and the final state that it has to reach and calculates the trajectory that the UAV must track to reach the final state in the assigned amount of time.

The trajectories are a sequence of desired positions, velocities and accelerations that the quadrotor has to reach. These desired states are given with a fixed rate to the controller described in Sec. 2.2.1.

Furthermore this module is continuously replanning the trajectories in order to compensate errors related to wrong final conditions or wrong trajectory tracking.

This module will be discussed extensively in Chapter 5.

# Chapter 3

## Base detection and tracking

One part of the work is focused on the state estimation of the moving platform. This is necessary in order to have a good prediction of the final state that the quadrotor must reach to land on the moving car.

With the method described in this section, every time we detect the platform, we can estimate its position, orientation and velocity in the world coordinate frame. Using this information as initial condition we can predict where the platform will be in  $t$  seconds.

An EKF [23] is design in order to have the most reliable value of the state of the platform.

Kalman filtering consists in an algorithm that uses a series of noisy measurements observed over time and, using a model of the dynamics of the system, produces estimates of unknown variables. This filter uses Bayesian inference and estimates a joint probability distribution over the variables for each time frame.

The algorithm works in a two-step process:

- in the prediction step, the KF produces estimates of the current state variables, along with their uncertainties, based on a model of the system:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k; \quad (3.1)$$

- once the outcome of the next measurement is observed

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k, \quad (3.2)$$

these estimates are updated using a weighted average: the greater the weight the higher is the certainty of the measurement.

In Eq. (3.1) and Eq. (3.2)

- $\mathbf{x}_k$  is the state to be estimated;
- $\mathbf{z}_k$  the measure;

- $\mathbf{u}_k$  the control input;
- $f$  the state dynamics;
- $h$  the measure model;
- $\mathbf{w}_k$  and  $\mathbf{v}_k$  the process and observation noises. These random variables are assumed to be multivariate Gaussian distribution with zero mean and covariances  $\mathbf{Q}_k$  and  $\mathbf{R}_k$  respectively.

In the EKF formulation the state transition  $f$  and observation models  $h$  do not need to be linear functions of the state but may instead be differentiable functions.

The algorithm is recursive and can run in real time, using only the current input measurements and the previously calculated state with its uncertainty matrix. The KF does not require any assumption that the errors are Gaussian. However, the filter yields to the exact conditional probability estimate in the special case that all errors can be modeled as Gaussian distributions.

In the following we summarize the mathematical stages we have to perform in order to calculate the final estimate, in the case in which we have discrete time prediction and update models.

### Initialization:

$$\hat{\mathbf{x}}_{0|0} = \mathbf{x}_0, \quad (3.3a)$$

$$\mathbf{P}_{0|0} = P_0. \quad (3.3b)$$

### Prediction step:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k), \quad (3.4a)$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1} \mathbf{P}_{k-1|k-1} \mathbf{F}_{k-1}^\top + \mathbf{Q}_k, \quad (3.4b)$$

where the state transition matrix is defined to be the following Jacobians:

$$\mathbf{F}_{k-1} = \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k}. \quad (3.5)$$

### Update step:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top (\mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k)^{-1}, \quad (3.6a)$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k (\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})), \quad (3.6b)$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}, \quad (3.6c)$$

where the observation matrix is defined to be the following Jacobian:

$$\mathbf{H}_k = \left. \frac{\partial h}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}}. \quad (3.7)$$

### 3.1 Prediction update: non-holonomic model

The platform is considered as a car and simulated with the non-holonomic model Fig. 3.1 and Eq. 3.8.

In this model, the state is defined as  $\mathbf{x} = (x, y, z, \theta, v_{tan}, \phi)$ : it corresponds to the 3 position in a space  $(x, y, z)$ , the yaw angle of the platform ( $\theta$ ) with respect to the world frame, the forward velocity ( $v_{tan}$ ), and the angle of the front wheels ( $\phi$ ). The system depends on a parameter  $L$  that corresponds to the distance between the front and the back wheels.

In this model the control input are the change in velocity  $u_1 = \dot{v}_{tan}$  and in the angle of curvature  $u_2 = \dot{\phi}$ .

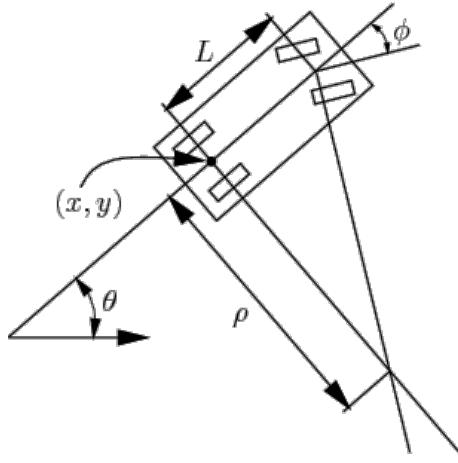


Figure 3.1: Non-holonomic model

The equation of motion in continuous time are:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \\ \dot{v}_{tan} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v_{tan} \cos(\theta) \\ v_{tan} \sin(\theta) \\ 0 \\ \frac{v_{tan}}{L} \tan(\phi) \\ u_1 \\ u_2 \end{bmatrix} = f(\mathbf{x}, \mathbf{u}) \quad (3.8)$$

It is possible to discretize these dynamics with first order finite difference:

$$\dot{\mathbf{x}} \approx \frac{\mathbf{x}(t_k) - \mathbf{x}(t_{k-1})}{t_k - t_{k-1}} = \frac{\mathbf{x}_k - \mathbf{x}_{k-1}}{dt} \approx f(\mathbf{x}_{k-1}, \mathbf{u}_k). \quad (3.9)$$

This leads to:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_k \\ v_{tan,k} \\ \phi_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + dt(v_{tan,k-1} \cos(\theta_{k-1})) \\ y_{k-1} + dt(v_{tan,k-1} \sin(\theta_{k-1})) \\ z_{k-1} \\ \theta_{k-1} + dt\left(\frac{v_{k-1}}{L} \tan(\phi_{k-1})\right) \\ v_{tan,k-1} + dt(u_{1,k}) \\ \phi_{k-1} + dt(u_{2,k}) \end{bmatrix} \quad (3.10)$$

In order to solve the former system a numerical solution is necessary. For this purpose we use a Runge-Kutta scheme [24].

In numerical analysis, the Runge-Kutta methods are a family of implicit and explicit iterative methods used in temporal discretization for the approximate solutions of ordinary differential equations.

The most widely known member of the Runge-Kutta family is generally referred to as RK4.

Let an initial value problem be specified as follows:

$$\begin{cases} \dot{\mathbf{y}} = f(\mathbf{y}, t) \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases}, \quad (3.11)$$

where  $\mathbf{y}$  is an unknown function of time  $t$ , which we want to approximate, and the function  $f$  and the data  $t_0$ ,  $\mathbf{y}_0$  are given.

Now we want to discretize the dynamics in the interval  $t \in [0, T]$ . We choose a step size  $h > 0$  such that the interval is divided in  $N$  sub-intervals. For each of these we define:

$$\begin{cases} \mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{6} (\alpha_1 + 2\alpha_2 + 2\alpha_3 + \alpha_4) \\ t_{k+1} = t_k + h \end{cases} \quad \forall k \in [0, N], \quad (3.12)$$

where:

$$\begin{cases} \alpha_1 = f(\mathbf{y}_k, t_k) \\ \alpha_2 = f\left(\mathbf{y}_k + \frac{h}{2}\alpha_1, t_k + \frac{h}{2}\right) \\ \alpha_3 = f\left(\mathbf{y}_k + \frac{h}{2}\alpha_2, t_k + \frac{h}{2}\right) \\ \alpha_4 = f(\mathbf{y}_k + h\alpha_3, t_k + h) \end{cases} \quad (3.13)$$

With this algorithm we calculate  $\mathbf{y}_{k+1}$ : the RK4 approximation of  $\mathbf{y}(t_{k+1})$ . As explained this quantity is determined by the sum of current value  $\mathbf{y}_k$  and the weighted average of four increments.

### 3.1.1 Straight and circular path

If we assume that the input  $u_{1,k}$  and  $u_{2,k}$  are equal to zero  $\forall k$  we can have three types of movement:

- the platform can be static ( $v_{tan,0} = 0$ );
- it can move in a straight line ( $v_{tan,0} \neq 0$  and  $\phi_0 = 0$ );
- it can move in a circle ( $v_{tan,0} \neq 0$  and  $\phi_0 \neq 0$ ).

A combination of these type of movements, even if they are really simple, leads to a large set of possible trajectories.

Furthermore, when we use this model in the prediction step of the EKF, we consider that it is corrupted by Gaussian noise. This way a modification in the control inputs  $u_{1,k}$  and  $u_{2,k}$  can be part of the model as far as it is slow and smooth.

### 3.1.2 Infinity shape path

As described in Chapter 1.2, the moving platform moves in an infinity-shape path described in Fig. 1.1.

We need to describe in a mathematical way this shape in order to use this information when we are estimating the state of the platform and to understand the right moment to perform the landing maneuver.

From the specification of the challenge:

- the car is moving with constant velocity  $v_{tan}$  along the path;
- the radius of the circumferences that forms the trajectory is  $\rho_8 = 20$  m;
- the path makes a cross in the middle that creates 4 angles of  $\frac{\pi}{2}$ .

It easy to see that the shape can be seen as a combination of a cross and two circles. If we consider the world coordinate system, this path can be located in any position with any orientation in the  $x, y$  plane. To describe mathematically the trajectory, we consider that the center of the cross is in the origin of the world coordinate frame. Then, if we want to model the real world path, we apply a rigid transformation in the  $x, y$  plane.

The cross is simply defined as the union between the two line:

$$y = x \tag{3.14a}$$

$$y = -x , \tag{3.14b}$$

while the two circles can be modeled as:

$$y^2 + (x - x_0)^2 = \rho_8^2 \tag{3.15a}$$

$$y^2 + (x + x_0)^2 = \rho_8^2 \tag{3.15b}$$

It is easy to see from Fig. 3.2 that if we want the intersections between the cross and the two circumferences to be exactly in 4 points we have to choose:

$$x_0 = \frac{\sqrt{2}}{2} \rho_8 \quad (3.16)$$

This corresponds to the 4 intersections coordinate:

$$\left( \frac{\rho_8 \sqrt{2}}{2}, \frac{\rho_8 \sqrt{2}}{2} \right); \left( \frac{\rho_8 \sqrt{2}}{2}, -\frac{\rho_8 \sqrt{2}}{2} \right); \left( -\frac{\rho_8 \sqrt{2}}{2}, -\frac{\rho_8 \sqrt{2}}{2} \right); \left( -\frac{\rho_8 \sqrt{2}}{2}, \frac{\rho_8 \sqrt{2}}{2} \right).$$

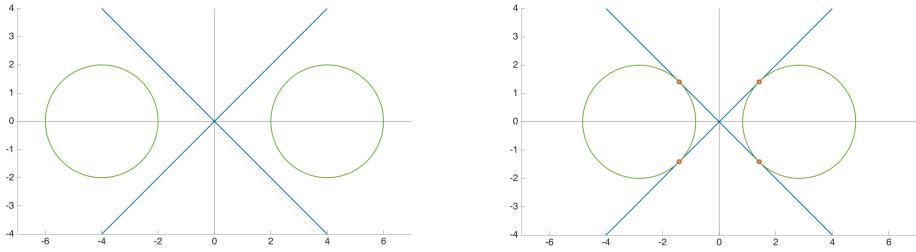


Figure 3.2: How to construct the infinity-shape path

The easiest way to describe this path is to define how the angle  $\theta$  is changing along the trajectory. If we travel over the two circumferences the intersections correspond to angles  $\theta = \pm \frac{3\pi}{4}$ .

Now it is obvious to see that the path is symmetric and it can be divided in 4 parts and, describing how the angle is changing in one of this section, the whole trajectory is defined.

We can observe that if  $x$  is the space traveled along the infinity path:

$$\theta(x) = \begin{cases} -\frac{x}{\rho_8} & x \in \left[0, \frac{3\pi}{4} \rho_8\right] \\ -\frac{3\pi}{4} & x \in \left[\frac{3\pi}{4} \rho_8, \frac{3\pi}{4} \rho_8 + \rho_8\right] \end{cases} \quad (3.17)$$

This function defines a quarter of the trajectory in function of the radius  $\rho_8$  of the path. Figure 3.3b shows the shape of this quarter of trajectory.

It is now possible to use it to generate the entire trajectory  $(x(t), y(t))$  described in Fig. 3.3c.

We know that the length of the path is

$$l = 4 \left( \frac{3\pi}{4} \rho_8 + \rho_8 \right), \quad (3.18)$$

and given the constant velocity  $v_{tan}$  we can calculate the time to complete the trajectory as:

$$T = \frac{l}{v_{tan}}. \quad (3.19)$$

Also, it is simple to define  $\theta(t)$  just stretching or shrinking  $\theta(x)$ . So we can now define:

$$\begin{cases} \dot{x} = v_{tan} \cos(\theta(t)) \\ \dot{y} = v_{tan} \sin(\theta(t)) \end{cases} \quad (3.20)$$

Finally, we also need the discretized version obtained just using forward Euler approximation:

$$\begin{cases} x_k = x_{k-1} + dt(v_{tan,k-1} \cos(\theta_{k-1})) \\ y_k = y_{k-1} + dt(v_{tan,k-1} \sin(\theta_{k-1})) \end{cases} \quad (3.21)$$

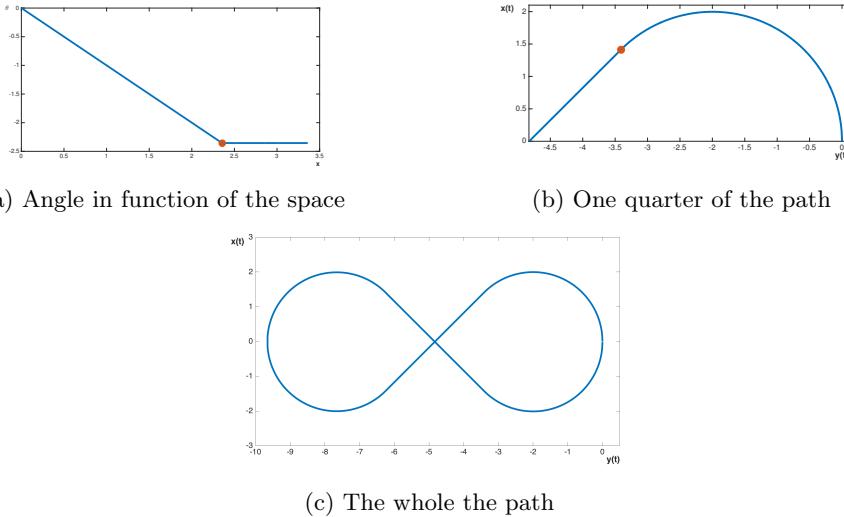


Figure 3.3: The parametrization of the path

From these calculations we can conclude that the final trajectory of the platform is only a composition of a linear and a circular movement (for a given amount of time). So the model described in Sec. 3.1.1 can be used also to express this trajectory.

## 3.2 Measurement update

From Eq. (3.10) we have the variables that describe the state of the moving car. We have to be able to measure some of these components in order to perform the second step of our EKF.

We use a down looking camera to identify the moving platform and to estimate its position and orientation. At this point, knowing the position of the camera

in the real world we can measure:

$$\mathbf{z}_k = h(\mathbf{x}_k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_k \\ v_{tan,k} \\ \phi_k \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_k \\ \phi_k \end{bmatrix} \quad (3.22)$$

It corresponds to the 3 position in a space  $(x, y, z)$  and the yaw angle of the platform  $\theta$  with respect to the world frame.

In Chap. 4 we will see that in order to complete the task, the quadrotor goes through various stages. In these different phases the relative distance between camera and moving platform changes a lot, so we have to use different methods to measure  $\mathbf{z}$ :

- to be able to find the platform in the minimum amount of time, at the beginning, we need to inspect the area from a very high altitude. From this height we can see only a few features of the moving car and then the pose estimation is really noisy. Furthermore, we do not have any assumption on the initial condition of the platform, but we just know the magnitude of constant forward velocity  $|v_{tan}|$ . Therefore, we do not know in advance if at a certain time  $t$  the car is moving on a straight line or in a curve. This renders necessary to observe the moving platform for several seconds in order to determine on which part of the track it is.
- After knowing the type of movement and a rough pose estimation of the moving car, we can use these information to improve our state estimation: getting close to the platform without loosing the tracking, a more precise measure is available (base on tag detection), and filtering the measurements with the correct theoretical model of the movement.

### 3.2.1 Platform pose estimation at high altitudes

To find the car, we assume that the platform is the only white square moving on the arena.

Based on this assumption, we analyze the images from the camera to find a moving white square and calculate its optical flow to predict its future position. To find the base we perform the following steps:

- thresholding the image in order to find the white blobs;
- finding all the close shapes in the image;
- selecting only the shapes having:
  - 4 edges;
  - convex contour;
  - angles between edges close to  $\frac{\pi}{2}$ .

At this point, we have the position of the four corners of the squares in the image.

Now we calculate the optical flow of these points through the sequence of images and we track only the points that are moving with a velocity comparable to the one known  $v_{tan}$ .

The optical flow methods [25] calculate the motion of each pixel between two subsequent images. For a 2D case, a pixel at location  $(x, y, t)$  with intensity  $I(x, y, t)$  moves by  $\Delta x, \Delta y$  and  $\Delta t$  between the two image frames. To solve this problem, the core assumption is the brightness constancy constraint:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \quad (3.23)$$

Assuming the movement to be small, the image constraint at  $I(x, y, t)$  can be approximated with its first order Taylor Series:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \simeq I(x, y, t) + \frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t. \quad (3.24)$$

It follows that:

$$\frac{\partial I}{\partial x} \frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y} \frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t} \frac{\Delta t}{\Delta t} = 0, \quad (3.25)$$

which results in:

$$I_x V_x + I_y V_y + I_t = 0, \quad (3.26)$$

where  $V_x, V_y$  are the  $x$  and  $y$  components of the velocity, or optical flow, of  $I(x, y, t)$ , and  $I_x, I_y, I_t$  are the derivatives of the image at  $(x, y, t)$  in the corresponding directions.

Thus in a compact form:

$$\nabla I^T \cdot \vec{V} = -I_t \quad (3.27)$$

This is a single equation in two unknowns and cannot be solved as such. This is known as the aperture problem of the optical flow algorithms. To find the optical flow another set of equations is needed, given by some additional constraint. All optical flow methods introduce additional conditions for estimating the actual flow.

In our implementation we use the Lucas-Kanade method [26]. This method assumes that the displacement of the image contents between two nearby frames is small and approximately constant within a neighborhood of the point  $p$  under consideration. Thus the optical flow equation can be assumed to hold for all pixels within a window centered at  $p$ . Namely, the local image flow vector  $(V_x, V_y)$  must satisfy:

$$\left\{ \begin{array}{l} I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2) \\ \vdots \\ I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n) \end{array} \right. \quad (3.28)$$

Where  $q_1, q_2, \dots, q_n$  are the pixels inside the window, and  $I_x(q_i), I_y(q_i), I_t(q_i)$  are the partial derivatives of the image  $I$  with respect to position  $x, y$  and time  $t$ , evaluated at the point  $q_i$  and at the current time.

These equations can be written in matrix form  $\mathbf{A}v = b$ , where

$$\mathbf{A} = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix} \quad (3.29)$$

This system has more equations than unknowns and thus it is usually over-determined. The Lucas-Kanade method obtains solution by using the least squares method:

$$\mathbf{A}^T \mathbf{A} \mathbf{v} = \mathbf{A}^T \mathbf{b} \quad (3.30a)$$

$$\mathbf{v} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} \quad (3.30b)$$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_y(q_i) I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix} \quad (3.31)$$

With this method we can track the interesting points, namely the platform corners, from frame to frame and calculate the direction and velocity of their movement. We need now a method to convert this position in the image into the correspondent pose in the real world.

### From images to real world

After tracking the platform in the images, we have to find its position in the 3D real world. This position is calculated using the pinhole model of the camera [27]:

$$\omega \mathbf{m} = \mathbf{A}[\mathbf{R}|\mathbf{t}] \mathbf{M} \quad (3.32a)$$

$$\omega \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3.32b)$$

Where:

- $\mathbf{m}$ : is the homogeneous coordinate of the point in the image expressed in pixel  $(u, v, 1)$ ;

- $\mathbf{M}$ : is the homogeneous coordinate of the correspondent 3D point in the world coordinate frame ( $X, Y, Z, 1$ );
- $\mathbf{A}$ : is the camera matrix, or the matrix of intrinsic parameters. It is composed by the focal lengths  $f_x, f_y$  and the principal point  $c_x, c_y$ ;
- $[\mathbf{R}|\mathbf{t}]$ : is the joint rotation-translation matrix, or matrix of extrinsic parameters. It expresses the camera motion around the static scene. This matrix denotes the pose of the camera in the world frame. In particular, we have to notice that the position  $\mathbf{C}$  of the camera expressed in world coordinates is  $\mathbf{C} = -\mathbf{R}^{-1}\mathbf{t} = -\mathbf{R}^T\mathbf{t}$ .

We can calculate the depth of the platform using the known dimension of the base: given the length  $l_w$  of the square in the real world and the average dimension of the edges in the image  $l_i$ , we can calculate the depth with respect to the camera frame

$$z = \frac{l_w f}{l_i} \quad (3.33)$$

To calculate the dimension  $l_i$  we need at least 3 corner of the base and we calculate all the pairwise distances between the corners (c.f. Fig. 3.4):

- if we have 4 corners there are 6 different distances: 4 of which equal to  $l_i$  and  $2\sqrt{2}l_i$  (Fig. 3.4a );
- if we have 3 corners there are 3 different distances: 2 of which equal to  $l_i$  and  $1\sqrt{2}l_i$  (Fig. 3.4b ).

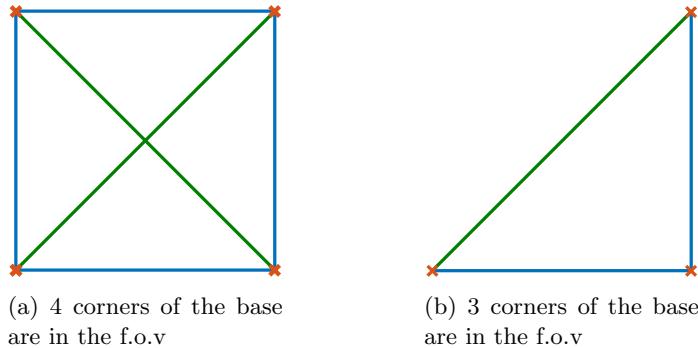


Figure 3.4: Model of the square platform detected on the image. Red crosses corner detected. Blue lines edges with length  $l_i$ . Green lines edges with length  $\sqrt{2}l_i$

Although this approximation is not accurate when we see the platform with a camera not perpendicular to the base, we use it to have a rough approximation of the height in this first phase. Since the distance between camera and platform is very high, this assumption can hold.

If this depth  $z$  is nonzero, we can solve the system of equation (3.32b) to find

an unique solution using the following equivalent equations:

$$x = z \frac{u - c_x}{f_x} \quad (3.34a)$$

$$y = z \frac{v - c_y}{f_y} \quad (3.34b)$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \mathbf{t} \quad (3.34c)$$

As previously said, this method is not very accurate, because we are assuming that the platform surface and the image plane are parallel. A better method to find the position of the platform, without the approximation of the depth  $z$ , is to resolve a Perspective-n-Point problem [28] that estimates the pose of a camera given a set of  $n$  3D points in the world and their corresponding 2D projections in the image. This method finds the pose that minimize the reprojection error of the 3D points in the image plane.

The main issue is that, to solve this problem, without ambiguity, the minimum number of points is 4, and sometimes we can track only 3 corners of the base. Therefore when all the 4 points are available we solve the correspondent PnP problem to find a better estimation of the base position, otherwise we use the former method.

At this point, with this algorithm, we can track the points of interest from frame to frame, the calculate direction, the velocity and the correspondent 3D pose of the car.

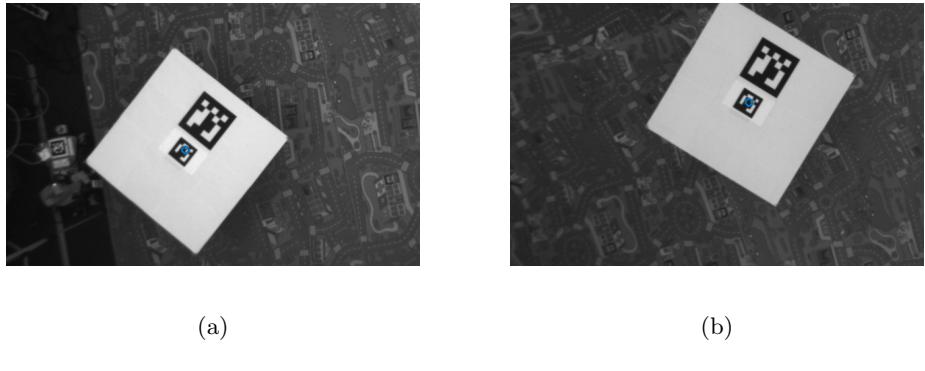


Figure 3.5: Real world example. The center of the platform is calculated detecting the white square and its corners. In figure (a) all the 4 corners of the base are visible in the image, while in figure (b) only 3 are in the f.o.v. . With the algorithm described in both cases we can calculate the position of the center.

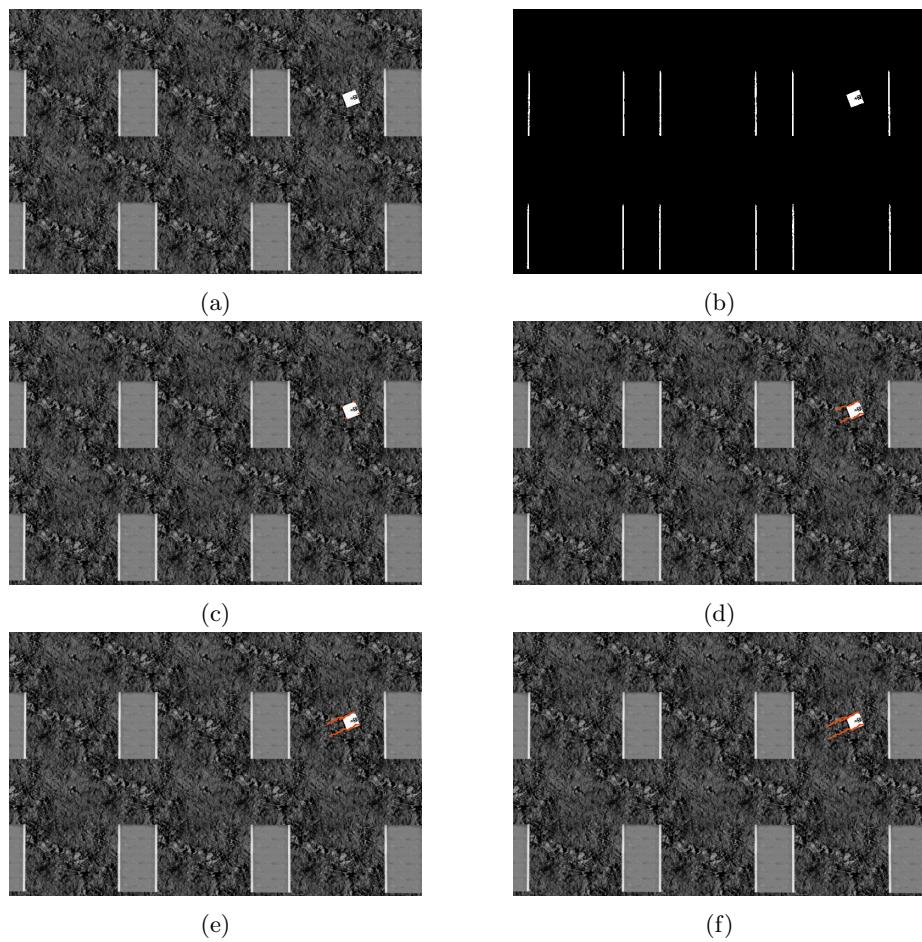


Figure 3.6: A sequence of images where the moving car is detected and tracked. The first image (a) is the original image. Then (b) is the thresholding. Then all the subsequent images (c)-(f) where the corners of the platform are tracked.

### 3.2.2 Platform pose estimation at low altitudes

When the quadrotor is close to the landing platform more features can be seen from the camera. In the final challenge, described in Chap. 1.2, the platform will be as depicted in Fig. 1.2, while for the first testing another design is considered, in order to use preexisting algorithms that allow pose estimation with respect to the camera.

The platform we are using is decorated with Augmented-Reality-Tags [29]. AR-Tags are planar markers used in computer vision to calculate, in real time and with high precision, the camera pose relative to the physical square. These markers are widely used for augmented reality: while a camera is taking a video with the marker in the f.o.v., in the images the square is replaced by any kind of objects. To reduce the sensitivity to lightning conditions and camera settings planar marker systems typically use bitonal markers (black and white). Therefore, there is no need to identify shades of gray, and the decision made per pixel is reduced to a threshold decision.

The temporary platform design used during this prototyping phase is depicted in Fig. 3.7: the markers consist of a black square with a pattern in the interior to allow an unique identification.

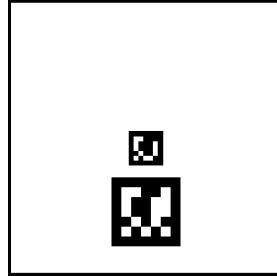
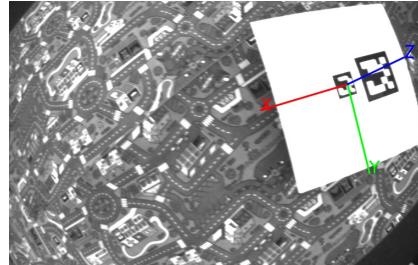


Figure 3.7: Design of the platform used in this work.

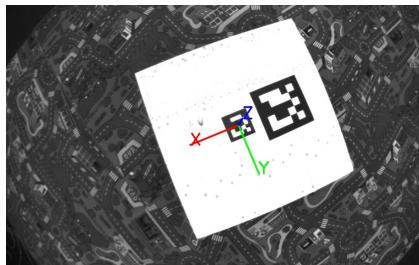
There are several methods to detect and calculate the pose of the markers. Some methods (as ARToolKit [30]), use a fixed global threshold to detect squares, but these methods are very sensitive to varying lighting conditions. On the other hand, other algorithms (as ARTag [31]), use an edge based approach, so one does not need to deal with thresholds under different illumination conditions and the algorithm can cope with broken sides and missing corners up to a certain extent. Both algorithms find in the image the contour of the marker, then the four corners of every potential marker are used to calculate a homography in order to remove the perspective distortion, solving a Perspective-n-Point problem [28]. Once the internal pattern of a marker is brought to a canonical front view one can sample a grid of  $N \times N$  (typically  $5 \times 5$  or  $6 \times 6$ ) points in order to understand the code related to the tag identified, and the orientation of the tag.



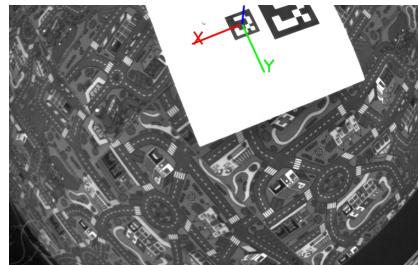
(a) If we are far from the moving platform we have to use the big tag to identify the base.



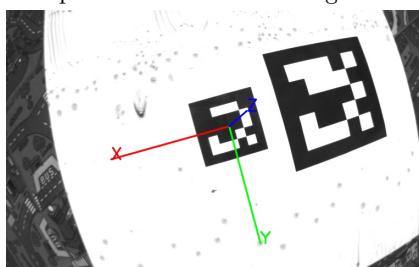
(b) Only when the bigger square is inside the FOV we can detect the center of the base correctly.



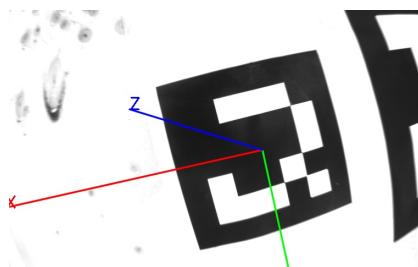
(c) When both the tags are visible we use both the information to have the best position of the master tag.



(d) Even if we lose one or more tag of the board, we still have the pose estimation of the center.



(e) The landing maneuver is performed to be finished over the central tag. So while we are landing the bigger tag is no more in the f.o.v.



(f) At the end, only the central tag is entirely on the f.o.v., so this tag must be little in order to have the possibility to track it until the very end.

Figure 3.8: A sequence of images where the AR-Tag over the base is detected. The coordinate system related to the moving platform has its origin on the master tag. The landing is performed over this tag.

### 3.2.3 Covariance Estimation

In the practical implementation of the Kalman Filter it is crucial to find a good estimate of the noise covariance matrices  $Q_k$  and  $R_k$  for the prediction and the measurement steps.

When a manual tuning is required, these matrices are considered diagonal, such as each component of the state vector is corrupted by a Gaussian processes that is independent of all the other coordinates. It is easy to give a physical interpretation to the components of the diagonal, so it is easy to find meaningful values for them.

Equations (3.1) depicts the general matrix formulation of the system corrupted by a multivariable Gaussian noise  $\mathbf{w}_k$ , but if we consider the covariance matrix  $Q_k$  to be diagonal we can split the equation into:

$$\begin{bmatrix} \dot{x}_k^1 \\ \dot{x}_k^2 \\ \vdots \\ \dot{x}_k^n \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}_{k-1}, \mathbf{u}_k) \\ f_2(\mathbf{x}_{k-1}, \mathbf{u}_k) \\ \vdots \\ f_n(\mathbf{x}_{k-1}, \mathbf{u}_k) \end{bmatrix} + \begin{bmatrix} w_k^1 \\ w_k^2 \\ \vdots \\ w_k^n \end{bmatrix}, \quad (3.35)$$

where  $w_k^i$  is a scalar Gaussian random variable with variance  $q_k^i$ .

This variance can now be directly related to the error that is computed when the variable is predicted with the theoretical model.

For the error update Eq. (3.2) the idea is the same:

$$\begin{bmatrix} z_k^1 \\ z_k^2 \\ \vdots \\ z_k^m \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_{k-1}) \\ h_2(\mathbf{x}_{k-1}) \\ \vdots \\ h_m(\mathbf{x}_{k-1}) \end{bmatrix} + \begin{bmatrix} v_k^1 \\ v_k^2 \\ \vdots \\ v_k^m \end{bmatrix} \quad (3.36)$$

with  $v_k^i$  scalar Gaussian noise with variance  $r_k^i$ .

If the values, used in the update step are not directly measured, but derive from other quantities, to calculate their covariance we should propagate the uncertainty: in Eq. (3.2)  $\mathbf{z}_k = h(\mathbf{x}_k)$  is not a direct measurement of  $\mathbf{x}_k$  but is a function of other  $\gamma_k$ , such as:

$$h(\mathbf{x}_k) = g(\gamma_k). \quad (3.37)$$

Then we can easily estimate the uncertainty that we have computed during the observation of  $\gamma_k$ , but in the EKF we need the correspondent error for the measures  $g(\gamma_k)$ .

In the linear case

$$g(\gamma_k) = \mathbf{A}\gamma_k \quad (3.38)$$

The covariance matrix  $\Sigma_g$  of  $g$  is related to  $\Sigma_\gamma$ , the covariance of the variable  $\gamma$ , by the equation:

$$\text{Cov}(g) = \text{Cov}(\mathbf{A}\gamma_k) = \mathbf{A}\text{Cov}(\gamma_k)\mathbf{A}^\top \quad (3.39\text{a})$$

$$\Sigma_g = \mathbf{A}\Sigma_\gamma\mathbf{A}^\top \quad (3.39\text{b})$$

If the function  $g$  is a set of non-linear combination of the variables  $\gamma_i$ , it must be linearized by approximation to a first-order Taylor series expansion:

$$g_i(\gamma_k) \approx g_i(\tilde{\gamma}_k) + \sum_j^n \frac{\partial g_i}{\partial \gamma_j} \Big|_{\tilde{\gamma}_k^j} (\gamma_k^j - \tilde{\gamma}_k^j), \quad (3.40)$$

where  $\frac{\partial g_i}{\partial \gamma_j} \Big|_{\tilde{\gamma}_k^j}$  denotes the partial derivative of  $g_i$  with respect to the  $j$ -th variable, evaluated at the measured component  $\tilde{\gamma}_k^j$ .

In matrix notation, the first-order Taylor series expansion is:

$$g(\gamma_k) \approx g(\tilde{\gamma}_k) + J \Big|_{\tilde{\gamma}_k} (\gamma_k - \tilde{\gamma}_k), \quad (3.41)$$

where  $J$  is the Jacobian matrix.

Since  $g(\tilde{\gamma}_k)$  is a constant, it does not contribute to the error on  $g$ , so the propagation of the error can be approximated with the linear case where  $A = J$ :

$$\Sigma_g \approx \mathbf{J}\Sigma_\gamma\mathbf{J}^\top \quad (3.42)$$

In our case, the update step is defined in Eq. (3.22), and it is computed with the methods described in the previous section. More specifically, we do not have a direct measure of the 3D position and angle  $\theta$ , but they derive from the measurement of the 2D position of the pixel that corresponds to the corners of the platform. To estimate the final variance of the measurements used in the update step of the EKF, we must start from the error computed in the image, and propagate the covariance through the functions we apply, to finally find the uncertainty of the 3D pose used. Given the function  $g : \mathbb{R}^{6 \times 6} \rightarrow \mathbb{R}^{2 \times 2}$ , that converts the real world coordinate in image coordinate, and its Jacobian matrix  $J \in \mathbb{R}^{2 \times 6}$  to calculate the covariance of the final pose estimate,  $\Sigma_{RW} \in \mathbb{R}^{6 \times 6}$ , from the covariance of the image position  $\Sigma_I \in \mathbb{R}^{2 \times 2}$ , we need to invert Eq. (3.42):

$$\Sigma_{RW} = (\mathbf{J}^\top \Sigma_I \mathbf{J})^{-1} \quad (3.43)$$

In our implementation the function  $g$  from image pixel  $(u, v)$  to 3D coordinate  $(x, y, z, \theta)$  is given by a composition of more functions:

- the main calculation from coordinates in the image to 6DoF pose is done using the OpenCV function *solvePnP* [32] that returns a pose expressed as 3D position  $(x, y, z)$  and the orientation expressed in the Rodrigues convection [33];
- then we convert the Rodrigues angles into a rotation matrix;
- finally the rotation matrix into roll-pitch-yaw notation.

To propagate the uncertainty through the composition of these functions, we have to calculate the Jacobian of this composition:

$$J_{solvePnP} = J_0 = \left[ \frac{\partial g}{\partial Rodrigues}, \frac{\partial g}{\partial xyzpos} \right] \quad (3.44a)$$

$$J_{RodriguesToR} = J_1 = \frac{\partial Rodrigues}{\partial R}, \quad (3.44b)$$

$$J_{RToEuler} = J_2 = \frac{\partial R}{\partial euler}, \quad (3.44c)$$

$$J_{Final} = J = \left[ \frac{\partial g}{\partial euler}, \frac{\partial g}{\partial xyzpos} \right] = J_0 \begin{bmatrix} J_1 J_2 & 0 \\ 0 & I \end{bmatrix} \quad (3.44d)$$

At this point applying Eq. (3.43) with the final Jacobian  $J_{Final}$  Eq. (3.44d), we have the covariance we need in the EKF.

# Chapter 4

## State machine

This section describes the module that, based on the state estimation of the UAV and of the moving platform, decides in which state the framework is, and which is the desired state that the quadrotor must reach in order to complete the mission.

This module implements a state machine and the flow diagram can be seen in Fig. 4.1.

It consists of 5 main parts in each of which the MAV has to complete a particular task in order to proceed with the successive stage. The task is defined by a precise final state that the quad must reach, and this final condition is considered reached when the position of the quad is inside a sphere of radius  $\rho_{reached}$  around the final position.

In the following we describe in detail all these stages and explain the computation that we perform in order to decide where the UAV must go and when a particular stage is considered concluded.

### 4.1 Takeoff and searching for the base

In this stage the quadrotor starts from a position near to the ground and has to explore a given area from high altitude until the platform is found.

At the beginning, the quad hovers close to the ground, then a vertical takeoff is performed until it reaches a given altitude  $h$ . This vertical maneuver is performed anytime the pipeline fails and we have to restart the state machine.

Given the area that must be explored to find the target (in our case it will be the arena in which the platform can move, see Fig. 1.1), we calculate a list of way-points the UAV must reach in order to span the whole area.

As the quadrotor moves, the camera collects information from a large section of the area and the searching task of the target can be performed faster.

There are many ways to sample the way-points to explore the area, in our case

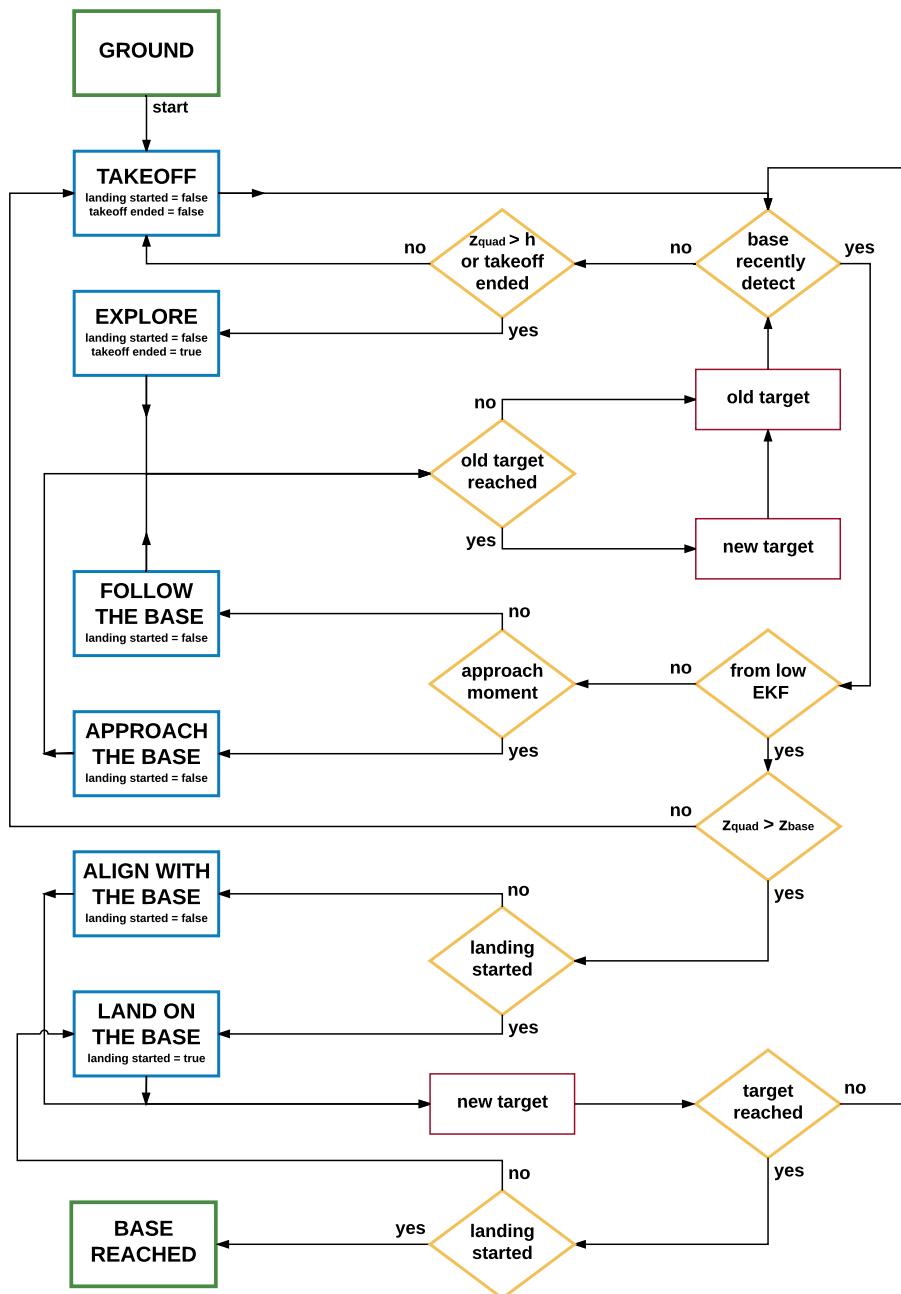


Figure 4.1: Landing state machine flow chart.

we try to maximize the probability to find the moving platform so we are moving from one side to the other, along the main axis of the infinity-shape path.

As soon as the moving platform is found, the state machine proceeds with the next stage.

## 4.2 Following the base

In this stage the quadrotor has to follow the moving platform until we identify the right moment it has to start the landing maneuvers.

The MAV moves at high altitude, reaching the desire points given by the previous stage. As soon as an estimation of the target state is available, the quad begins to follow the platform and performs the following computations in order to complete the task of this stage.

### 4.2.1 Understand type of movement

From the challenge description in Sec. 1.1, we know that the car moves in a shape composed by straight lines and circumference sectors. We need to understand, at a given time, in which part the platform is: this information is important in order to calculate properly where the platform will be in  $t$  seconds. Also, we want to perform the landing maneuver, when the platform is moving on a straight line.

To understand the trajectory of the moving base, we collect all the estimated positions of the base coming from the previous module, described in Chap. 3, and we perform a linear regression on the last  $n$  estimations. The platform moves in a straight line if the linear regression is a good approximation of the data trends, otherwise it driving on a curve.

We have a series of  $n$  points, each of these is consider as a pair of coordinates  $(x_i, y_i)$ , and we are searching for the best-fit line that can describe the data as a linear function:

$$y = mx + q \quad (4.1)$$

In our case, there are no real dependent and independent variables so we perform the following analysis considering before the coordinates  $y_i$  as dependent, then solve the dual problem with  $x_i$ , and finally peaking the fit with smaller approximation error.

We want find the best parameters  $m$  and  $q$ , and to do so we need to have some measures of quality to optimize. Unless all our  $n$  points are already in a perfect line there will be an error between the value predicted by the line, and the observed dependent variable:

$$e_i = y_i - (mx_i + q) \quad (4.2)$$

These differences are called residuals and what we want is to find the model that minimizes:

$$(m^*, q^*) = \operatorname{argmin} \sum_{i=1}^n e_i^2 \quad (4.3)$$

The model we find is the Least Squares Fit of the data. We define also the cumulative residual as:

$$e_{tot} = \sqrt{\sum_{i=1}^n e_i^2} \quad (4.4)$$

The parameters  $m$  and  $q$  of the model are found where  $e_{tot}^2$  is minimized. To do so, the following first order condition must hold:

$$\frac{\partial e_{tot}^2}{\partial m} = 0, \quad (4.5a)$$

$$\frac{\partial e_{tot}^2}{\partial q} = 0. \quad (4.5b)$$

It is easy to demonstrate that the solution of Eq. (4.5b) is:

$$m^* = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2}, \quad (4.6a)$$

$$q^* = \frac{\sum_{i=1}^n y_i}{n} - m \frac{\sum_{i=1}^n x_i}{n}. \quad (4.6b)$$

The platform moves on a straight line if the cumulative residual  $e_{tot}$  is below a threshold  $th_{line}$ , while if the error is above  $th_{curve}$  the base is traveling the circumference. If  $th_{line} \leq e_{tot} \leq th_{curve}$  then the type of movement cannot be determinate, and we assume that the platform keeps moving with the same style found before.

To have a good interpretation of the data it is important to decide the three parameters  $n$ ,  $t_{line}$ ,  $t_{curve}$  correctly:

- The first parameter  $n$  is the number of samples to consider when we perform the linear regression. We chose it in order to consider poses that are along a curve with length:

$$l_{curve} = \frac{\rho_8 \pi}{4}. \quad (4.7)$$

We know the forward constant velocity of the car  $v_{tan}$ , so we can calculate the time it takes the platform to perform this curve:

$$t_{curve} = \frac{l_{curve}}{v_{tan}}. \quad (4.8)$$

When we receive a pose at time  $t_i$  we store it and we perform the linear regression with all the data stored in the interval  $[t_i - t_{curve}, t_i]$ .

- The threshold parameters are calculated considering that each measure is corrupted by an additive Gaussian noise with 0 mean and  $\sigma_e^2$  variance:

$$\tilde{y}_i = \mathcal{N}(y_i, \sigma_e^2). \quad (4.9)$$

When we perform the linear regression on the measured data, the average residual square is

$$\begin{aligned} <\tilde{e}_i^2> &= <(\tilde{y}_i - (mx_i + q))^2> \\ &= <\tilde{y}_i^2 - 2\tilde{y}_i(mx_i + q) + (mx_i + q)^2> \\ &= <\tilde{y}_i^2> - 2<\tilde{y}_i>(mx_i + q) + (mx_i + q)^2 \quad (4.10) \\ &= \sigma_e^2 + y_i^2 - 2y_i(mx_i + q) + (mx_i + q)^2 \\ &= \sigma_e^2 + e_i^2 \end{aligned}$$

Now we have to distinguish if the total approximation error is related to a straight line or a curve section:

- when we perform the linear regression on linear data the theoretical data are distributed as:

$$\begin{cases} x_i = x_i \\ y_i = ax_i + b \end{cases}, \quad (4.11)$$

so the theoretical residual, calculated using Eq. (4.2), is:

$$e_i = ax_i + b - (mx_i + q). \quad (4.12)$$

When we try to calculate the model parameter  $m^*$  and  $q^*$  with Eq. (4.6b) the result leads to:

$$\begin{cases} m^* = a \\ q^* = b \end{cases}. \quad (4.13)$$

So, obviously, the theoretical residual squares is:

$$e_i^2 = 0, \quad (4.14)$$

and the average residual squares on the measured data is:

$$<\tilde{e}_i^2> = \sigma_e^2. \quad (4.15)$$

The parameter  $th_{line}$  is then:

$$th_{line} = \sqrt{\sum_{i=1}^n \tilde{e}_i^2} = \sqrt{\sum_{i=1}^n \sigma_e^2} = \sigma_e \sqrt{n}. \quad (4.16)$$

- When we perform the linear regression on data along a circumference arch with radius  $\rho$  and angles  $\theta_i \in [\theta_1, \theta_2]$ , the theoretical data are distributed as:

$$\begin{cases} x_i = \rho \cos \theta_i \\ y_i = \rho \sin \theta_i, \end{cases} \quad (4.17)$$

so the theoretical residual, calculated using Eq. (4.2), is:

$$e_i = \rho \sin \theta_i - (m\rho \cos \theta_i + q) \quad (4.18)$$

To find  $m^*$  and  $q^*$  we use Eq. (4.6b), but we want a general approximation of these values. To do so, we have to consider all the sums in the equations as integrals, using the relation:

$$\lim_{n \rightarrow \infty} \frac{b-a}{n} \sum_{i=0}^n f(x_i) = \int_a^b f(x) dx \quad (4.19)$$

$$\sum_{i=0}^n f(x_i) \simeq \frac{n}{b-a} \int_a^b f(x) dx \quad (4.20)$$

So now if we calculate this approximation for our values we have:

$$\begin{aligned} \sum_{i=1}^n x_i y_i &= \sum_{i=1}^n \rho^2 \cos \theta_i \sin \theta_i \\ &\simeq \frac{n}{\theta_2 - \theta_1} \rho^2 \int_{\theta_1}^{\theta_2} \cos x \sin x dx \\ &= \frac{n}{\theta_2 - \theta_1} \frac{\rho^2}{2} \left[ -\cos^2 x \right]_{\theta_1}^{\theta_2} \end{aligned} \quad (4.21)$$

$$\begin{aligned} \sum_{i=1}^n x_i &= \sum_{i=1}^n \rho \cos \theta_i \\ &\simeq \frac{n}{\theta_2 - \theta_1} \rho \int_{\theta_1}^{\theta_2} \cos x dx \\ &= \frac{n}{\theta_2 - \theta_1} \rho \left[ \sin x \right]_{\theta_1}^{\theta_2} \end{aligned} \quad (4.22)$$

$$\begin{aligned}
\sum_{i=1}^n y_i &= \sum_{i=1}^n \rho \sin \theta_i \\
&\simeq \frac{n}{\theta_2 - \theta_1} \rho \int_{\theta_1}^{\theta_2} \sin x dx \\
&= \frac{n}{\theta_2 - \theta_1} \rho \left[ -\cos x \right]_{\theta_1}^{\theta_2}
\end{aligned} \tag{4.23}$$

$$\begin{aligned}
\sum_{i=1}^n x_i^2 &= \sum_{i=1}^n \rho^2 \cos^2 \theta_i \\
&\simeq \frac{n}{\theta_2 - \theta_1} \rho^2 \int_{\theta_1}^{\theta_2} \cos^2 x dx \\
&= \frac{n}{\theta_2 - \theta_1} \frac{\rho^2}{2} \left[ x + \cos x \sin x \right]_{\theta_1}^{\theta_2}
\end{aligned} \tag{4.24}$$

In our case we consider pieces of curve with length  $l_{curve}$  defined in Eq. 4.7, that correspond to a circumference arch with:

$$\rho = \rho_8 \quad \theta_i \in \left[ 0, \frac{\pi}{4} \right] \tag{4.25}$$

We can now calculate the approximate values of  $m^*$  and  $q^*$  using Eq. (4.6b) and Eqs. (4.21) - (4.25):

$$\begin{aligned}
m^* &= \frac{n \rho_8^2 \frac{n}{\pi} - \rho_8 \frac{n 2 \sqrt{2}}{\pi} \rho_8 \frac{n 2 (2 - \sqrt{2})}{\pi}}{n \frac{n \rho_8^2 (2 + \pi)}{2\pi} - (\rho_8 \frac{n 2 \sqrt{2}}{\pi})^2} \\
&= \frac{2\pi - 16\sqrt{2} + 16}{\pi^2 + 2\pi - 16}
\end{aligned} \tag{4.26a}$$

$$\begin{aligned}
q^* &= \frac{\rho_8 \frac{n 2 (2 - \sqrt{2})}{\pi}}{n} - m \frac{\rho_8 \frac{n 2 \sqrt{2}}{\pi}}{n} \\
&= \rho_8 \frac{4 - 2\sqrt{2}(m + 1)}{\pi} = \rho_8 \bar{q}
\end{aligned} \tag{4.26b}$$

Now we have to calculate the theoretical residual square  $e_i^2$ , but in this case we can compute the algebraic average residual square  $\langle e_i^2 \rangle$ , using again the approximations (4.21) - (4.24), the values calculate in (4.26), and averaging over the  $n$  samples we consider:

$$\begin{aligned}
\langle e_i^2 \rangle &= \frac{1}{n} \sum_{i=1}^n \left( \rho_8 \sin \theta_i - (m \rho_8 \cos \theta_i + q) \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^n \rho_8^2 \xi = \rho_8^2 \xi
\end{aligned} \tag{4.27}$$

Finally we calculate

$$\langle \tilde{e}_i^2 \rangle = \langle e_i^2 \rangle + \sigma_e^2 = \rho^2 \xi + \sigma_e^2. \quad (4.28)$$

The parameter  $th_{curve}$  is then:

$$th_{curve} = \sqrt{\sum_{i=1}^n \tilde{e}_i^2} = \sqrt{\sum_{i=1}^n \rho^2 \xi + \sigma_e^2} = \sqrt{n}(\sigma_e + \rho \sqrt{\xi}). \quad (4.29)$$

Figure 4.2 shows the typical evolution of the total residual during this first stage: the different phases of linear and circular movement can be detect in the graph. Furthermore, the points of regime change can be seen both in Fig. 4.2 and in Fig. 4.3 in which also all the estimated positions of the base are plotted.

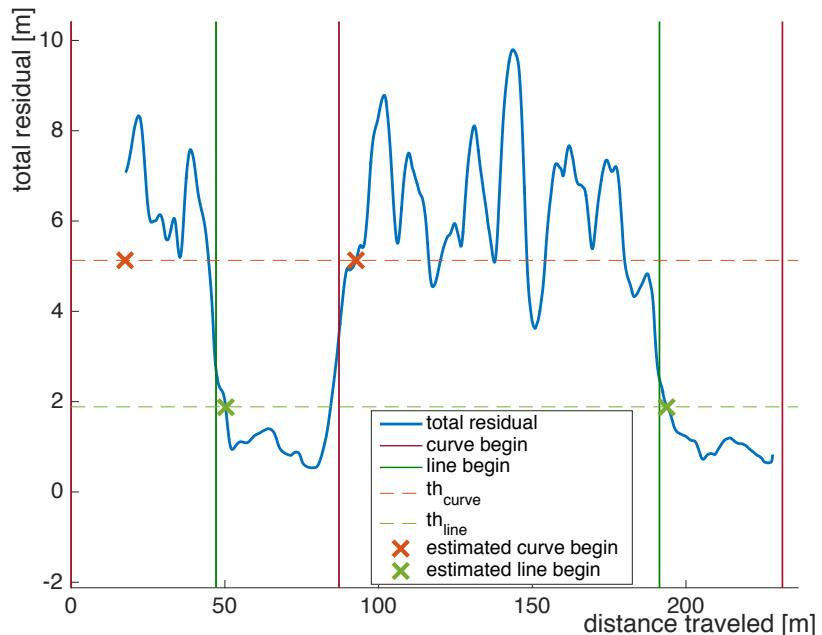


Figure 4.2: Evolution of the total residual during this first phase (in blue). The vertical lines are the real moments in which the car changes movement types: green a linear phase starts, red a circular phase begins. The horizontal lines are the thresholds for the detection of the two different phases. The crosses are the moments in which the algorithm understands the change.

We can see from these graphs that with the proposed method is possible to distinguish clearly the period of time in which the platform is traveling along a line or along a curve, and it is also finds the switching points with good accuracy. The major drawback of this method is that it is necessary an amount of time equal to  $t_{curve}$  to understand that the platform switched motion regime. As a matter of fact, to detect a straight line movement we need that all the positions, considered in the linear regression, lay on the line.

If  $th_{line}$  and  $th_{curve}$  are too close, it is always possible to consider a curve

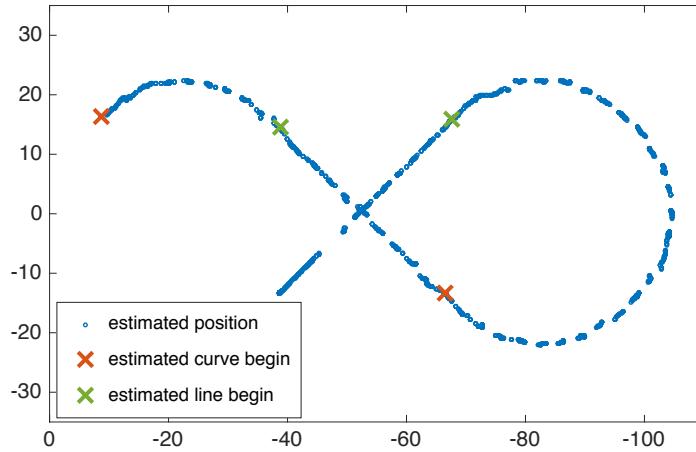


Figure 4.3: Map of the estimated positions of the platform in blue. The crosses are the moments in which the algorithm understands the change. Red crosses from line to curve. Green crosses from curve to line.

with longer length  $l_{curve}$ : this will increase the latter threshold with respect to the former, but will also increase the time  $t_{curve}$  to understand the type of movement.

#### 4.2.2 Calculate future positions of the moving platform

If the platform regime of movement at a specific time is known, we can estimate where it will be after  $t_s$  seconds and proceed with the following stages when it starts a straight portion of the trajectory.

Thanks to the algorithm described before, we can estimate that if at time  $t_0$  the car is at position  $(x_0, y_0)$  with a direction angle of  $\theta_0$  and forward velocity of  $v_{tan}$ , at time  $t_1 = t_0 + t_s$  the car will be at position  $(x_1, y_1)$  with an angle  $\theta_1$ , and it has traveled  $v_{tan}t_s$ .

- When no regime is found (at the beginning) or when a line movement is detected, the predicted state is:

$$\begin{cases} x_1 = x_0 + v_{tan}t_s \cos \theta_0 \\ y_1 = y_0 + v_{tan}t_s \sin \theta_0 \\ \theta_1 = \theta_0 \end{cases} . \quad (4.30)$$

- When a movement on the circumference is detected, we have to perform some calculations in order to find the final state of the platform.

First, we use the relation:

$$l_{curve} = v_{tan}t_s = \rho_8 |\beta_s|, \quad (4.31)$$

where  $\beta_s$  is the angle spanned by the platform in  $t_s$  seconds:

$$|\beta_s| = \frac{v_{tan} t_s}{\rho_8}. \quad (4.32)$$

The final angle will be:

$$\theta_1 = \theta_0 + \beta_s. \quad (4.33)$$

Referring to Fig. 4.4 we can calculate that the segment connecting  $(x_0, y_0)$  and  $(x_1, y_1)$  has:

- direction  $\theta_{chord}$  found as bisection between  $\theta_0$  and  $\theta_1$ :

$$\theta_{chord} = \theta_0 + \frac{\theta_0 + \theta_1}{2}; \quad (4.34)$$

- length  $l_{chord}$ , found with the chord theorem:

$$l_{chord} = 2\rho_8 \sin \frac{|\beta_s|}{2}. \quad (4.35)$$

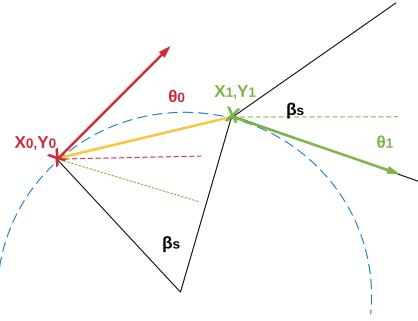


Figure 4.4: In red, the position of initial state at time  $t_0$ . In green, the final estimate state at time  $t_1$ . In yellow, the chord between the two states with length  $l_{chord}$  and orientation  $\theta_{chord}$ .

Now we have all the elements to calculate the final point  $(x_1, y_1)$ , but in order to properly find it we have to resolve another last problem. Both  $\beta_s$  and  $-\beta_s$  span a curve of length  $v_{tan} t_s$ , and due to the symmetry of our trajectory is impossible to know beforehand which angle is the right one. What we can do is calculate both the two possible final states using Eqs. (4.32), (4.34) and (4.35):

$$\begin{cases} x_1^a &= x_0 + l_{chord} \cos \left( \theta_0 + \frac{|\beta_s|}{2} \right) \\ y_1^a &= y_0 + l_{chord} \sin \left( \theta_0 + \frac{|\beta_s|}{2} \right) , \\ \theta_1^a &= \theta_0 + |\beta_s| \end{cases} \quad (4.36)$$

$$\begin{cases} x_1^b &= x_0 + l_{chord} \cos \left( \theta_0 - \frac{|\beta_s|}{2} \right) \\ y_1^b &= y_0 + l_{chord} \sin \left( \theta_0 - \frac{|\beta_s|}{2} \right) . \\ \theta_1^b &= \theta_0 - |\beta_s| \end{cases} \quad (4.37)$$

In order to understand which one is the correct state, we can calculate the distance between the two possible final points and a point of the trajectory estimated at time  $t_{-\alpha} < t_0$ . The state with smaller distance will be the right final state, because the wrong one leads to a position further away.

Figure 4.5 summarizes the passages we perform to find the right final state explained above.

At this point we can use the predict position of the platform to control the quadrotor following the base.

The image 4.6 shows the points in which the algorithm calculates where the quadrotor should go in order to follow the moving car. It is noticeable the subdivision of point calculates with the linear model (green stars) and with the circular one (red stars).

### 4.2.3 Select moment to land

At this point the quadrotor is following the base, we have now to understand when the right moment to proceed with the other stages is arrived.

The right moment to start the landing maneuver is at the start of a line segment:

- if we detect the base and we understand that it is moving in the circumference we cannot land, we have to follow the base and waiting when we will detect a change in the regime from curve to line, at this point we can proceed with the following stages.

Proceed with the landing at this point can be risky if the platform is moving fast: as a matter of facts we know that the length of the straight line is  $2\rho_8$  but we understand that the platform is moving straight after  $l_{curve} = \frac{pi}{4}$  after it finished the curve, so the quad has just

$$t_{landing} = \frac{\rho_8(2 - \frac{pi}{4})}{v_{tan}}$$

seconds to perform the land with the platform moving in line. If the velocity of the platform is too high this time could not be sufficient.

- if we first detect the base and we understand that it is moving in a straight line we should not land, because we do not know when it actually started the line, so it can be almost at the end of it, and we do not have time to perform the entire landing maneuver.

What we do is following the car and waiting when it changes movement regime, from line to curve. At this point we can calculate where the next change point, from curve to line, will be and starting the landing at that point.

In this case we will have more time to complete the landing, because the quadrotor starts the maneuver at the begin of the line, no after  $l_{curve}$ , and, indeed, it can proceed with the first stage of the landing even a

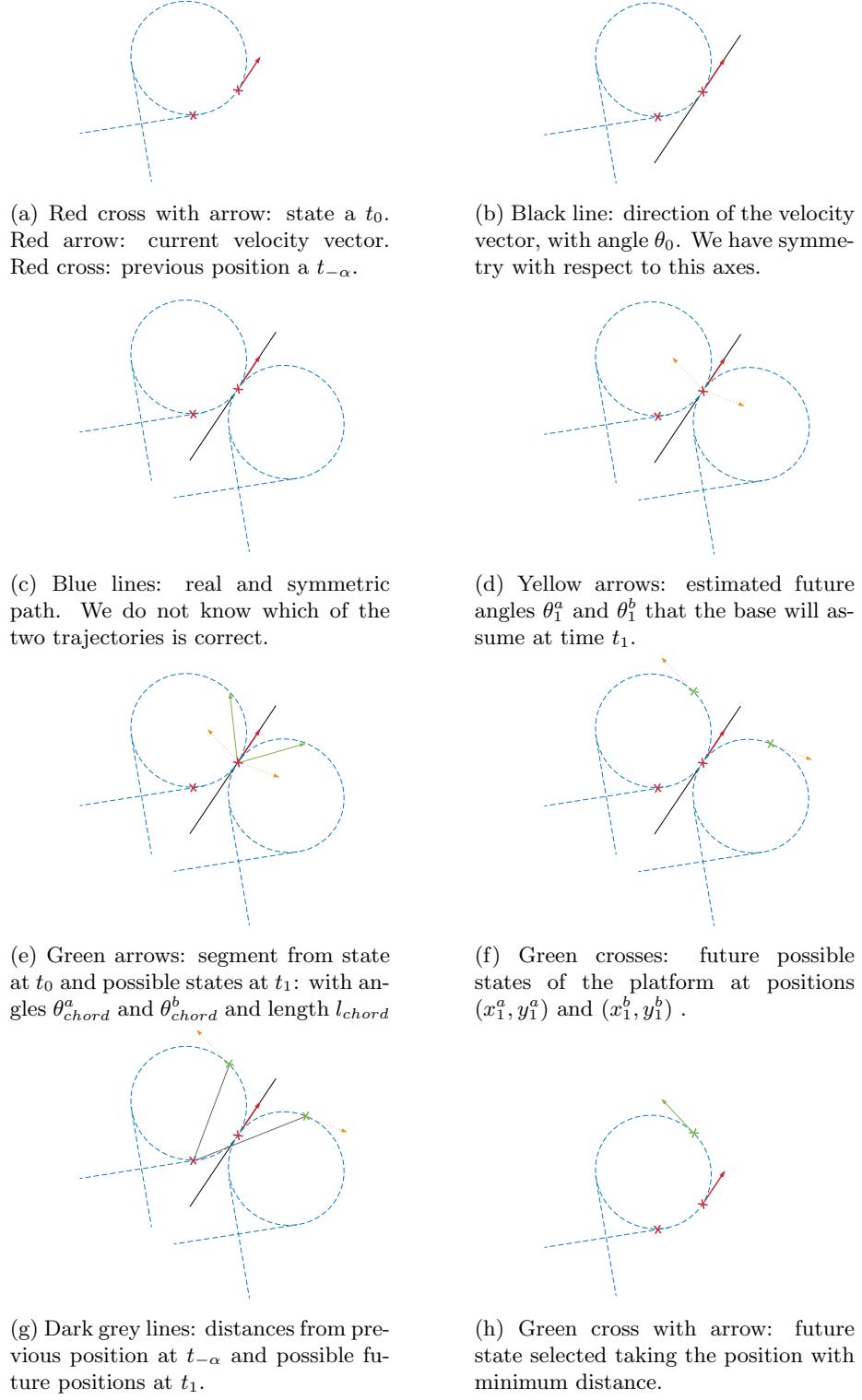


Figure 4.5: The sequence of passages computed in order to select the future position when the platform is moving on the circumference.

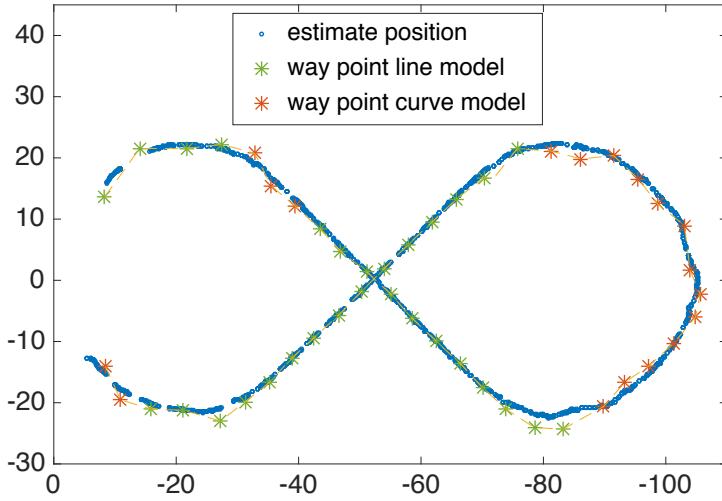


Figure 4.6: Map of the estimated positions of the platform in blue. Stars positions in which the quadrotor should go to following the base until a proper moment to proceed with the following stage is detected. The green points are calculated with the linear model while the red ones with the circular.

little before the start of the segment, so:

$$t_{landing} \geq \frac{2\rho_8}{v_{tan}}$$

In order to calculate where the future changing point will be, we must perform some computations:

- we know the orientation  $\theta_{line}$  of the straight line just finished: the platform just changed from line regime to curve and we have saved the inclination  $m$  of the best linear approximation found while it was moving in line.
- given the point of change between line and curve, the future point will be in the circumference after an angle of  $|\frac{3\pi}{2}|$ .
- from equations (4.34) (4.35) we know that the segment connecting the change point and the future intersection point has length  $\sqrt{2}\rho_8$  and angle  $\theta_{line} \pm \frac{3\pi}{4}$
- we can apply the same method described before to find the two possible intersection points:

$$\begin{cases} x_{intersection}^a = x_{changing} + \sqrt{2}\rho_8 \cos\left(\theta_{line} + \frac{3\pi}{4}\right) \\ y_{intersection}^a = y_{changing} + \sqrt{2}\rho_8 \sin\left(\theta_{line} + \frac{3\pi}{4}\right) \\ \theta_{intersection}^a = \theta_{line} + \frac{3\pi}{2} \end{cases}$$

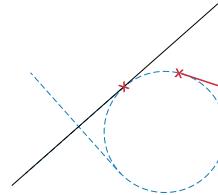
$$\begin{cases} x_{intersection}^b = x_{changing} + \sqrt{2}\rho_8 \cos\left(\theta_{line} - \frac{3\pi}{4}\right) \\ y_{intersection}^b = y_{changing} + \sqrt{2}\rho_8 \sin\left(\theta_{line} - \frac{3\pi}{4}\right) \\ \theta_{intersection}^b = \theta_{line} - \frac{3\pi}{2} \end{cases}$$

and select the right one with minimum distance with the current estimate position of the platform.

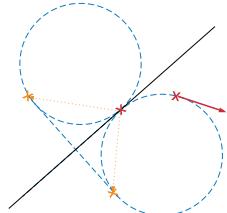
The images 4.7 summarize all the passages we perform to find the right intersection point just explained.



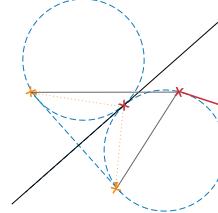
(a) Red cross with arrow: state a  $t_0$ . Red arrow: current velocity vector. Red cross: changing point from line to curve.



(b) Black line: direction of the line sector just finished. The direction is taken as the slope of the best linear fit found in the previous regime.



(c) Blue lines: real and symmetric path. We do not know which of the two trajectories is correct. Yellow crosses: in both the path we can calculate the future intersection point.



(d) Dark grey lines: distances from current position and the two possible future intersections. Both are eligible because of the symmetry of the trajectory.



(e) Yellow cross with arrow: future intersection point selected taking the position with minimum distance from the current state.

Figure 4.7: The sequence of passages computed in order to select the future intersection point where the platform will start the movement in line.

At this point the quad is keeping following the moving platform, but as soon as the base is near the future changing point, the quad can proceed with the next stage.

The image 4.8 shows where the algorithm calculates the future changing points (yellow crosses), based on the current one (red crosses). When the platform is in a neighborhood of these points for the UAV is the right moment to approach the base.

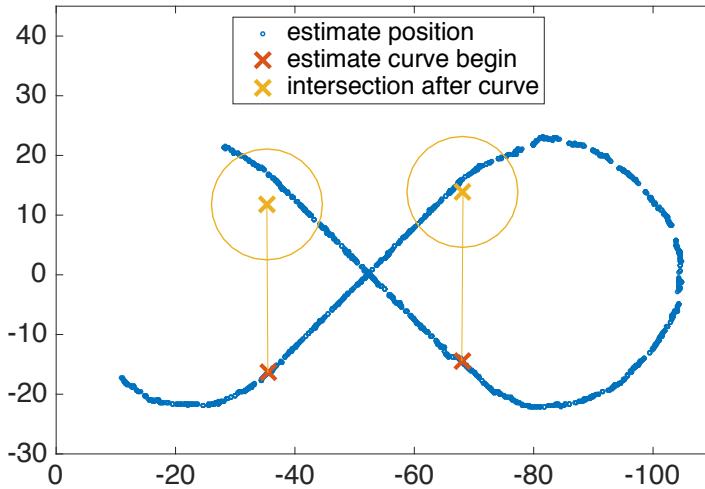


Figure 4.8: Map of the estimated positions of the platform in blue. Yellow crosses the position where the quadrotor should go in order to intersect the platform when is about to start a line phase.

### 4.3 Approaching the base

In this stage the quadrotor has to decrease its altitude keeping the platform in the fov until a better state estimation (from the low altitude EKF) is available.

The quadrotor is following the base and at high altitude. As soon as the platform start a straight line sector, the UAV must approach the platform reducing its altitude and keeping the target in the fov of the camera.

In this phase the desired final  $x, y$  coordinates of the quadrotor are calculate with the same equations of the previous stage when a line movement of the platform is detected (4.30). The main difference are about the  $z$  coordinate and the  $x, y$  final velocities that the quad has to assume.

We set the final velocity identical to estimate velocity of the base:

$$\begin{cases} vx_1 &= v_{tan} \cos \theta_0 \\ vy_1 &= v_{tan} \sin \theta_0 \end{cases} \quad (4.38)$$

While the final altitude:

$$z_1 = \alpha z_0 + (1 - \alpha) z_{0,target} \quad (4.39)$$

Where  $\alpha < 1$  is a parameter to be tuned in order to have a right balance between fastness of this stage and aggressiveness of the maneuver.

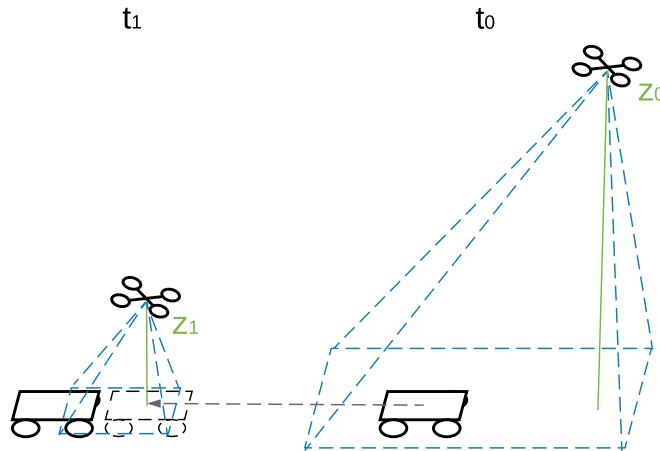
As a matter of fact if the quad approaches too quickly the target, it is very easy to lose the platform from the fov of the camera: the closer we are to the base the less area we can cover with the camera, so the more precise we must be in order to still have tracking of the target.

If we set the final state of the quad with the state estimation of the moving base we have from high altitude ( $\alpha = 0$ ) we are directly performing the landing on the base, but the UAV could approach a final position that is not the right one and so it can loose the platform. It is necessary to have some intermediate stages in which the quad is getting closer to the platform, so it can refines and correct the final target, but the area spanned by the camera is large enough to detect the base even if the quad is not in the right pose.

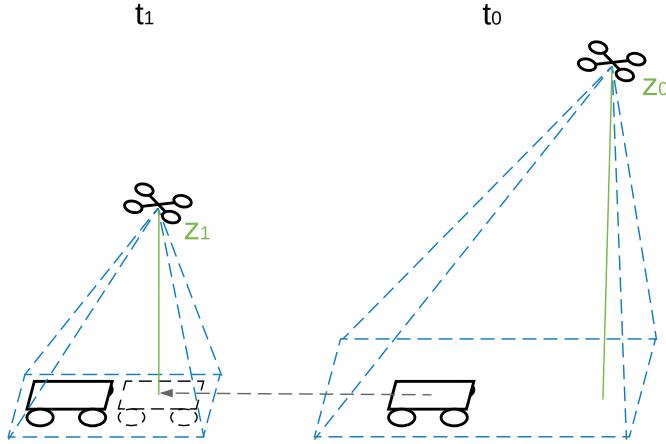
If we loose the platform, this stage fails, and we have to takeoff again until the platform is in the fov again.

The figure 4.9 summarize a situation in which approaching the platform too aggressively can lead to the loss of tracking, while if we have intermediate steps we can recover from the estimation error we had.

With this approach the platform can stay in the fov of the camera much easier: the quad is going ahead of the base, with a direction that is the same of the target, and correcting the estimate position of the moving platform at each step. As soon as we are close enough to have a more precise state estimation from the low altitude EKF we proceed with the next stage.



(a) If we set  $z_1$  too close to the platform we can loose the tracking because the position estimation of the moving base is not good enough from high altitude.



(b) With intermediate steps, instead, it is more difficult to loose the platform and we can correct the state estimation.

Figure 4.9: The scheme describe a situation in which intermediate steps, while approaching the moving platform from high altitude, are necessary to not loose the platform.

#### 4.4 Align with the base

In this stage the quadrotor is flying close to the platform and before proceeding with the landing maneuver it has to align its direction with the base to be in the best position to proceed with the final step.

When a state estimation of the base from the low altitude EKF is available the quad can rely on this information to align its movement with the base. The estimation is good enough to predict the future states of the moving platform even if we loose tracking. As a matter of fact in this stage we know that the platform is moving in a straight line, we know its initial position, direction and velocity, so it is easy to predict where it will be in  $t$  seconds.

Given the current state of the platform and of the UAV we can calculate the distance  $d$  between the two. This segment can be covered by the quadrotor with a maximum relative speed:

$$v_{max,rel} = v_{max,quad} - v_{base} \quad (4.40)$$

In a time:

$$t = \frac{d}{||v_{max,rel}||} \quad (4.41)$$

We can see from equation (4.40) that if the maximum velocity of the quadrotor is equal in magnitude and opposite direction with respect to the velocity of the platform the time  $t$  to reach the platform is infinite.

In this period of time we know that the platform is moving to a different place, and we have to predict where it will be in  $t$  seconds.

To predict the future position we are using the same model used in the EKF (3.10). With this model we do not just predict where the platform will be after  $t$  seconds, but we estimate its state for discrete moments in a window of time around  $t$ :  $[t - t_1, t + t_2]$ . In this way we have a set of possible future position that the quadrotor must be able to reach perfectly in time to intersect the platform. It is also possible to bring the quad ahead of the platform simply requiring to reach a state of the window in less time.

This window of future state estimate is used by the trajectory generator module to calculate different possible alternatives to reach the platform.

In this stage the final target of the quadrotor will be one of these odometry estimations with  $x, y$  position and velocity equal to the one of the platform, and  $z$  position set to a proper height  $h_{align}$ .

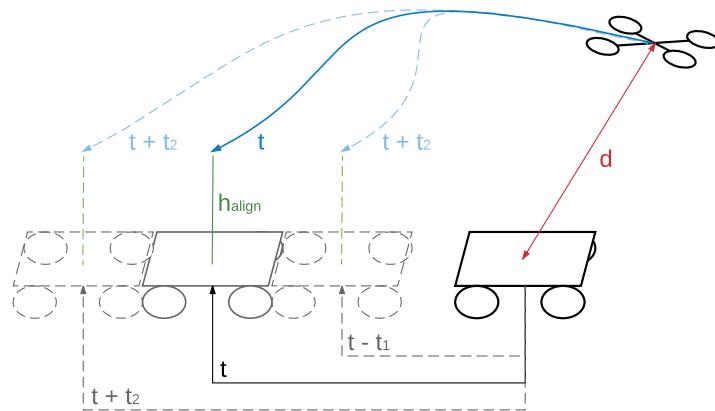


Figure 4.10: The scheme synthesizes the algorithm performed in this stage to find the possible final states in which the quadrotor should reach in order to intercept the base.

As soon as the quadrotor reaches the final target state, in which it is align with the moving platform, and the final target was in the fov recently, we proceed with the final stage.

## 4.5 Landing on the base

In this final stage the quadrotor is very close to the platform and is moving in the same direction. It has to decreases its altitude until it touches the platform and then switches off the motors in order to conclude the whole task.

The way to calculate the final target is equal to the previous stage of the state machine: we predict possibles intersection points that can be reachable by the

quadrotor, and the final states of the UAV will be one of these odometries. It is possible also to add a velocity and/or acceleration in the  $z$  direction in order to have a slightly more aggressive vertical landing. Furthermore because the visual odometry can fails when we are really close to the platform we introduce the possibility of a blind landing, in which we are not using the state estimation of the quadrotor to control the UAV, but we control the thrust in an open loop: when we are at  $h_{blind}$  over the platform we start to apply a thrust  $c_{quad}$  such that  $\|c_{quad}\| < g$  in order to decrease the altitude of the quadrotor until it touches the platform.

In detail the quad in this phase is moving in the  $z$  axis with the following kinematics:

$$z(t) = z_{quad,0} + v_{zquad,0}t + \frac{(c_{quad,z} - g)t}{2} \quad (4.42)$$

If the quad starts from  $z_{quad,0} = h_{blind}$  after a time  $t_{blind}$  we want  $z(t_{blind}) = 0$ . furthermore we consider the initial velocity in  $v_{zquad,0}$  really little, so it can be neglect. We can then easily calculate the time  $t_{blind}$ :

$$t_{blind} = \sqrt{\frac{2h_{blind}}{g - c_{quad,z}}} \quad (4.43)$$

We know that in this time  $t_{blind}$  the platform will changed its position, in particular it is moving in a straight line, so its coordinates will be :

$$\begin{cases} x_{base}(t_{blind}) = x_{base}(0) + v_{tan}t_{blind} \cos(\theta_{base}) \\ y_{base}(t_{blind}) = y_{base}(0) + v_{tan}t_{blind} \sin(\theta_{base}) \end{cases} \quad (4.44)$$

So we know that if we have selected the coordinates  $\tilde{x}, \tilde{y}$  as intersection points where we will start the blind landing over the platform, the quad must be at these coordinates  $t_{blind}$  seconds before the platform in order to properly land over it (notice that if  $h_{blind} \rightarrow 0$  also  $t_{blind} \rightarrow 0$  and so the quad arrives at the intersection point with the moving base).

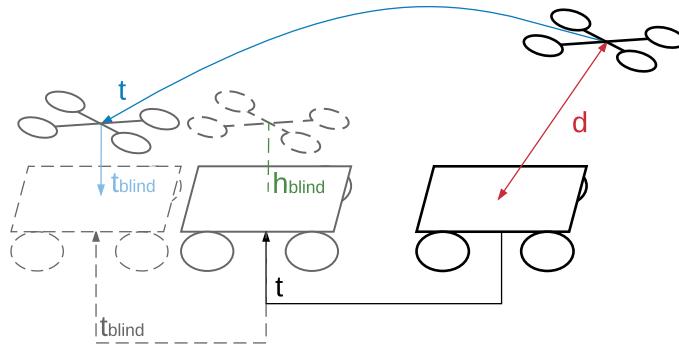


Figure 4.11: The scheme synthesizes the concept of the final blind landing.

In order to detect when the quad is touching the platform we check the data from the IMU.

This unit is given different measurements, among which the value of the linear accelerations along the 3 axes. Using this data we can calculate the magnitude of the acceleration and we know that when the UAV is hitting a surface this quantity is showing a big pick, so with a simple threshold on the acceleration norm we can detect when the quadrotor is landed.

This solution does not take in account that data from the IMU are usually corrupted by noise (see 5.1.6) and so we could confuse a noisy measurement with a bump.

To make this detection more robust we can filter the data with a low pass filter:

$$imu_{filt}(t_k) = (1 - e^{-\frac{t_k - t_{k-1}}{\tau_{imu}}}) imu_{raw}(t_k) + e^{-\frac{t_k - t_{k-1}}{\tau_{imu}}} imu_{filt}(t_{k-1}) \quad (4.45)$$

The filter eliminates the noise but it slows down the response to changes, so the detection of the bump is done with some delay: the parameter  $\tau_{imu}$  is deciding the cut frequency of the filter, so tuning this quantity can lead to a final filtered data with the right balance between smoothness and sensibility to changes.

These methods that uses the absolute value of the acceleration, do not take in consideration that the quadrotor could assume a very high acceleration while is performing a normal flight, and so this acceleration could exceed the threshold and be detected as a bump.

Another solution to have a robust and fast bump detector is to compare the raw data from the IMU with the filtered one: only the bump creates a big change that the filter cannot follow instantaneously, so the difference between the two version of the data will be very high only in this occasion.

The figure 4.12 shows the data used by the bump detector in order to find when the quad is touching the platform and switch off the motors.

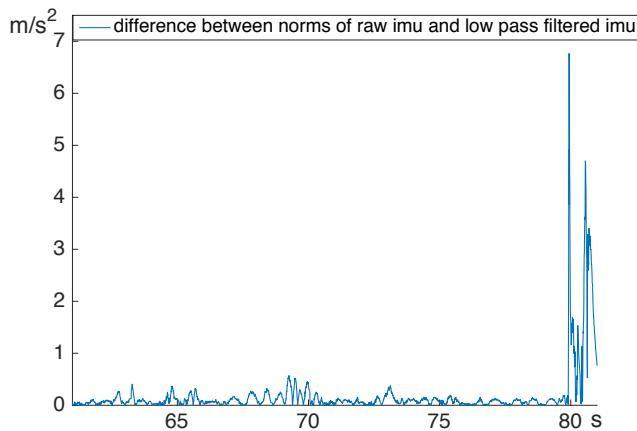


Figure 4.12: Data used by the bump detector: difference between the norms of the raw data from the imu and the filtered version of the same. The difference is growing really fast only when the UAV bumps on the surface.

If something goes wrong and in this phase the quadrotor reaches a  $z_{quad} < z_{base}$ , then the landing failed and we have to takeoff again.

## Chapter 5

# Trajectory generator

This section describes the module that computes the trajectory between the UAV current odometry estimation and a final desired target.

A trajectory is a sequence of desired states that leads the UAV from an initial condition at  $t = t_0$  to the desired final condition reached at  $t = T$ . In particular a desired state at a certain time  $t_i$  is defined as:

- $[x_{t_i,des}, y_{t_i,des}, z_{t_i,des}]$ : desired 3D position
- $[vx_{t_i,des}, vy_{t_i,des}, vz_{t_i,des}]$ : desired linear velocity
- $[ax_{t_i,des}, ay_{t_i,des}, az_{t_i,des}]$ : desired linear acceleration
- $[\psi_{t_i,des}]$ : desired yaw

The fact that the desired state of the quadrotor is completely defined by these quantities is because the quadrotor dynamics are differentially flat [34]: the states and the inputs can be written as algebraic functions of four flat outputs and their derivatives:  $[x, y, z, \psi]$ .

The initial desired state, for  $t_i = t_0$ , is given by the state estimation of the quad, while the final condition for  $t_i = T$  are given from the state machine module.

The final conditions can be of different types, and the calculation of the possible trajectories depends on it:

- During the first two stages of the state machine the final state is simply a pose in the world frame with zero velocity and acceleration. This module is calculating some trajectories from the initial state to this final states with different total times  $T_i$  and it is choosing the best one.  
The times  $T_i$  are depending on the distance between initial and final position and the average velocity that the quad should have during the flight.
- During the third stage the final state is a pose in the world frame with a velocity equal to the moving platform and zero acceleration. The module

is calculating the trajectories like in the previous point, so for different times  $T_i$  and picking the best option.

- In the other parts, in which the UAV has to align and land on the base, the state machine is given to the trajectory generator a set of possible final states with positions  $p_i$ , velocities  $v_i$  and times  $T_i$  to reach them. This module is calculating all the trajectories to reach all these possible final conditions in the correspondent time, and is choosing the best one.

Note that the choice of the final trajectory among all the possible calculates is done w.r.t a cost function that will be discuss later.

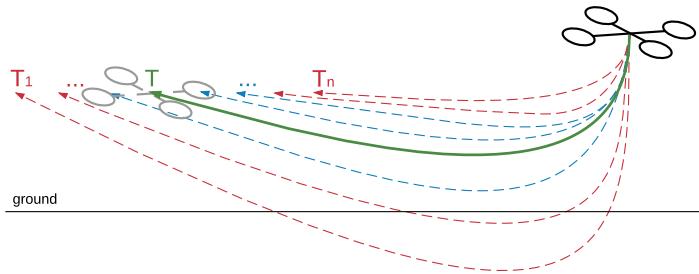


Figure 5.1: The scheme synthesizes the concept of multiple possible trajectory generated and then pick the best one. The red trajectory are unfeasible (state or input unfeasible), the green trajectory is the best solution found.

This module is constituted by two threads:

- The first thread is popping and publishing the top of a stack of desired states with rate  $r_{trj}$ .  
This state will be the input of the high controller module.
- The Second thread is:
  - receiving the initial and final conditions
  - checking if this two belong to the previous trajectory (within an error), and only if they do not, proceed with the following tasks
  - calculating the best trajectory
  - sampling the trajectory with a given rate  $r_{trj}$
  - substituting the desired states inside the stack of the first thread with the new ones sampled

In this module we utilize the trajectory planning approach described in [16] to generate thousands of trajectories per second (2ms each), and then choose the best one to follow. We are doing this calculation with frequent replanning in order to correct any errors related to the prediction of the final target or related to a displacement between desired state and actual state of the quadrotor, due to the not perfect tracking of the trajectory by the controller.

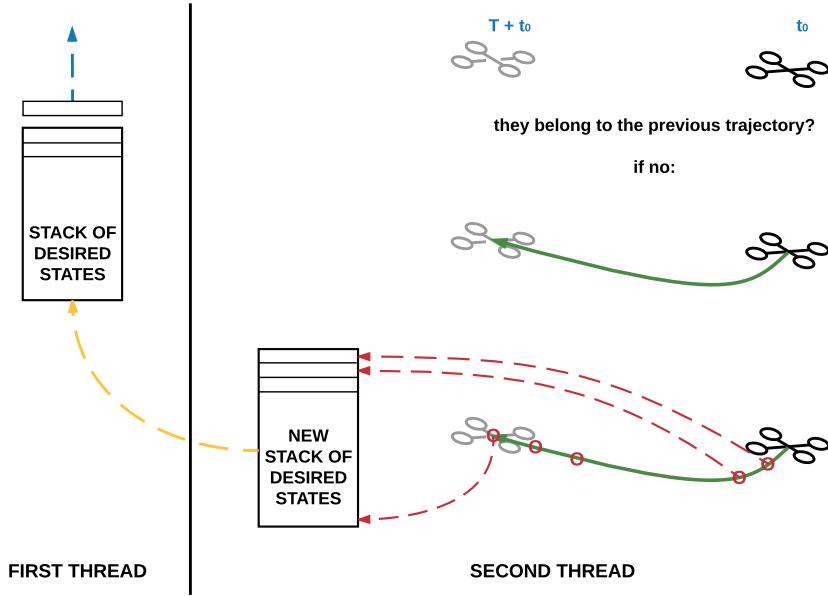


Figure 5.2: Scheme of tasks subdivision between the two threads.

## 5.1 Rapid Trajectory

The algorithm by Mark Mueller produces trajectories that are the result of an optimal control problem with the goal of computing a thrice differentiable trajectory which guides the quadrotor from an initial state (position, velocity, acceleration and yaw of the UAV) to a final state in a finite time  $T$ , while minimizing a cost function that can be considered as an upper bound on the average of a product of the inputs to the quadrotor system.

Furthermore, the final trajectory takes in account feasibility with input and space constraints.

### 5.1.1 Dynamic model

Starting from the classic simplified dynamic model of the quadrotor:

$$\begin{cases} \ddot{\mathbf{r}} = \mathbf{g} + \mathbf{R}_{WB}\mathbf{c} \\ \dot{\mathbf{R}}_{WB} = \mathbf{R}_{WB}\hat{\mathbf{w}}_{WB} \end{cases} \quad (5.1)$$

Where

$$\hat{\mathbf{w}}_{WB} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 0 \\ 0 \\ c \end{bmatrix} = \mathbf{e}_3 c \quad (5.2)$$

Where the system input are  $c$ , the total normalized thrust, and the angular rates  $\omega_1, \omega_2, \omega_3$ .

Now the goal is to express these inputs in function of the states and the jerk.

The input thrust  $c$  is computed by applying the norm to the position dynamics:

$$c^2 = \|\mathbf{c}\|^2 = \|\ddot{\mathbf{r}} - \mathbf{g}\|^2 \quad (5.3)$$

$$\begin{aligned} 2c\dot{c} &= 2(\ddot{\mathbf{r}} - \mathbf{g})^T \mathbf{j} = 2c\mathbf{e}_3^T \mathbf{R}_{WB}^T \mathbf{j} \\ \dot{c} &= \mathbf{e}_3^T \mathbf{R}_{WB}^T \mathbf{j} \end{aligned} \quad (5.4)$$

We can also define the position dynamic in terms of jerk:

$$\mathbf{j} = \ddot{\mathbf{r}} = \dot{\mathbf{R}}_{WB} \mathbf{c} + \mathbf{R}_{WB} \dot{\mathbf{c}} \quad (5.5)$$

Combining the two derivations, we can say that fixed  $\mathbf{j}$  and  $c$ , we define uniquely two components of the body rates:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = \frac{1}{c} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R}_{WB}^T \mathbf{j} \quad (5.6)$$

Using these equations the inputs of the system are defined with a degree of freedom in  $\omega_3$ .

The goal of this algorithm is to find the trajectory  $\mathbf{z}(t)$  for  $t \in [0, T]$ , consisting of the quadrotor position, velocity and acceleration, such that:

$$\mathbf{z}(t) = (\mathbf{r}(t), \dot{\mathbf{r}}(t), \ddot{\mathbf{r}}(t)) \in \mathbb{R}^9 \quad (5.7)$$

with given initial and final conditions  $\mathbf{z}(0)$  and  $\mathbf{z}(T)$ .

If we consider the system input to be the three-dimensional jerk, then we can decoupling the dynamics into three orthogonal inertial axes, and treating each axis as a triple integrator with jerk used as control input. The true control inputs  $c$  and  $\omega$  are then recovered from  $\mathbf{j}$  using equations 5.4 and 5.6.

### 5.1.2 Optimal control problem

The trajectory generation is rewritten as a discrete optimal control problem, with boundary conditions defined by the quadrotor initial and (desired) final states. The solution of this problem must minimize a cost function subject to some dynamics and satisfying state and inputs conditions.

As we said, the dynamics are split among the three decoupled axis, and for each axis the optimal control problem is solved independently.

For a single axis the problem is defined as following:  
find the sequence of control input  $j_k$  that minimizes:

$$J = \sum_{k=0}^{N-1} j_k^2 \quad (5.8)$$

subject to the dynamics:

$$j_k = \ddot{r}_k$$

$$z_k = \begin{bmatrix} r_k \\ \dot{r}_k \\ \ddot{r}_k \end{bmatrix} = \begin{bmatrix} 1 & dt & \frac{dt^2}{2} \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} z_{k-1} + \begin{bmatrix} \frac{dt^3}{6} \\ \frac{dt^2}{2} \\ dt \end{bmatrix} j_{k-1} \quad (5.9)$$

$$z_0 = z(0)$$

$$z_N = z(T)$$

and respecting the constraints:

$$A \begin{bmatrix} r_k \\ j_k \end{bmatrix} \leq b \quad (5.10)$$

The solution of this optimal control problem can be found in close form with Pontryagin's minimum principle. In the paper [16] are presented all the calculations to derive the solution.

The final result requires the evaluation of a single matrix that depends on the initial and final condition  $z(0)$   $z(T)$  and the total time  $T$ .

### 5.1.3 Cost function

The cost function selected is, considering the three axis together:

$$J = \sum_{k=0}^{N-1} \|j_k\|^2 \quad (5.11)$$

This cost function has been chosen because it can be interpreted as an upper bound for a product of the input (using equation 5.6):

$$c_k^2 \|\omega_k\|^2 = \left\| c_k \begin{bmatrix} \omega_{1,k} \\ \omega_{2,k} \end{bmatrix} \right\|^2 = \left\| c_k \frac{1}{c_k} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R}_{WB,k}^T \mathbf{j}_k \right\|^2 \leq \|j_k\|^2 \quad (5.12)$$

### 5.1.4 Constraints

The trajectory is feasible if  $c$  and  $\|\omega\|$  respect the following for all  $t$  of the trajectory:

$$\begin{aligned} 0 < c_{min} \leq c \leq c_{max} \\ \|\omega\| \leq \omega_{max} \end{aligned} \quad (5.13)$$

These constraints can be rewritten in term of the state and the jerk: for the thrust

$$\begin{aligned} c_{min}^2 \leq c^2 \leq c_{max}^2 \\ c_{min}^2 \leq \|\ddot{\mathbf{r}} - \mathbf{g}\|^2 \leq c_{max}^2 \end{aligned} \quad (5.14)$$

For the body rates

$$\|\omega\| = [\omega_1 \quad \omega_2] \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} + \omega_3^2 = [\omega_1 \quad \omega_2] \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} \leq \frac{1}{c} \|\mathbf{j}\| \leq \omega_{max} \quad (5.15)$$

in which we assume that  $\omega_3 = 0$

### 5.1.5 Feasibility check

A fast conservative check is applied to check the feasibility of the trajectory. For the thrust, from equation 5.14, we know that the trajectory is unfeasible if:

$$\max_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{M}_z g)^2 > c_{max}^2 \quad \forall i \in \{x, y, z\} \quad (5.16)$$

$$\min_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{M}_z g)^2 < c_{min}^2 \quad \forall i \in \{x, y, z\} \quad (5.17)$$

where  $\mathbb{M}_z$  is equal to 1 if we are considering the  $z$  axis otherwise it is 0.

On the other hand trajectory is surely feasible if:

$$\sum_i \max_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{M}_z g)^2 \leq c_{max}^2 \quad i \in \{x, y, z\} \quad (5.18)$$

$$\sum_i \min_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{M}_z g)^2 \geq c_{min}^2 \quad i \in \{x, y, z\} \quad (5.19)$$

If both these checks fails the trajectory is considered interminable.

For the body rates from equation 5.15 we know that the trajectory is feasible only if:

$$\frac{\sum_{i=1} \max_{k=[0,N]} j_{i,k}^2}{\sum_{i=1} \min_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{M}_z g)^2} \leq \omega_{max} \quad i \in \{x, y, z\} \quad (5.20)$$

If the trajectory is define indeterminable then the feasibility check are repeated separately in the two sub intervals  $[1, \frac{N}{2}], [\frac{N}{2} + 1, N]$ , iteratively. The check stops when all the subset are feasible, or one is unfesible, or if the subdivision has arrived at intervals smaller than a certain threshold.

### 5.1.6 Compute the acceleration

The rapid trajectory generator needs an initial and a final state. The initial state is always selected as the current position velocity and acceleration of the quadrotor. From the state estimate of MSF we have the first two information, while we have to find a way to estimate the acceleration.

There are several ways to make this estimation:

- IMU: the Inertial unit gives measurements of the 3D linear accelerations when the quad is moving. These measures are really noisy when the quadrotor is flying because the motors are introducing vibrations that are corrupting the data from this unit. So to be used it is necessary to filter the measure with a low pass:

$$a_{imu}(t_k) = \left(1 - e^{-\frac{t_k - t_{k-1}}{\tau_{a_{imu}}}}\right) a_{imu}(t_k) + e^{-\frac{t_k - t_{k-1}}{\tau_{a_{imu}}}} a_{imu}(t_{k-1}) \quad (5.21)$$

- Finite difference: Having two successive velocity estimation we can calculate the acceleration approximating the derivative of the velocity with a numerical finite difference

$$\dot{v}(t_k) \simeq \frac{v(t_k) - v(t_{k-1})}{t_k - t_{k-1}} \quad (5.22)$$

Also this method is really sensitive to high frequency noise, and the data must be filtered with a low pass filter:

$$a_{fd}(t_k) = \left(1 - e^{-\frac{t_k - t_{k-1}}{\tau_{fd}}}\right) \frac{v(t_k) - v(t_{k-1})}{t_k - t_{k-1}} + e^{-\frac{t_k - t_{k-1}}{\tau_{a_{fd}}}} a_{fd}(t_{k-1}) \quad (5.23)$$

- Thrust: from the equation of motion of the quadrotor we know that the acceleration of the UAV in a specific moment are completely described by the total thrust  $c$  applied and the rotation of the quadrotor  $\mathbf{R}_{WB}$ :

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \mathbf{R}_{WB} \begin{bmatrix} 0 \\ 0 \\ c \end{bmatrix} \quad (5.24)$$

And we also know that:

$$c = \frac{1}{m} \sum_{i=1}^4 f_i \quad (5.25)$$

where  $f_i$  is the thrust produced by the propeller  $i$ .

From the low level controller 2.4 we have these values and so we can calculate the acceleration vector.

It is important to notice that the information from the low level control are the desired thrust for each propeller  $\tilde{f}_i$ , not the actual one  $f_i$ . The real produced thrust can be calculated as  $\lambda_i \tilde{f}_i$  where  $\lambda_i$  is the rotor fitness coefficients.

In the final implementation we decided to use the thrust, that, even if shows some offset in the z direction (look section 6.5.1 for more details) w.r.t the other two, it seems more smooth and does not need a filtering.

## 5.2 Minimum snap trajectory

When the rapid trajectory algorithm fails to find a feasible trajectory, and no previous trajectories are available, we have to find another way to calculate the sequence of desired states.

If this is the case, we are using a minimum snap trajectory [35]. This type of trajectories are the solution of another optimization problem in which the inputs are expressed in function of the fourth derivative of the position: the snap.

The problem formulation uses a more complete dynamics of the quad w.r.t the jerk formulation 5.1 :

$$\begin{cases} \ddot{\mathbf{r}} = \mathbf{g} + \mathbf{R}_{WB}\mathbf{c} \\ \dot{\mathbf{R}}_{WB} = \mathbf{R}_{WB}\hat{\mathbf{w}}_{WB} \\ \dot{\mathbf{w}}_{WB} = J^{-1}(\boldsymbol{\tau} - \mathbf{w}_{WB} \times J\mathbf{w}_{WB}) \end{cases} \quad (5.26)$$

where  $\boldsymbol{\tau}$  are the torques acting on the body caused by the motor thrust 2.3.

Using these dynamics we need a derivative more in order to express the input in terms of the flat outputs. Because of that, the states must be described as position, velocity, acceleration and jerk.

In order to compute this type of trajectory we should calculate the initial jerk, but since it is difficult to estimate its value we set the initial and final jerks to be zero (even if this condition is not correct for the initial state), while the other values of the initial condition are calculated as in the previous section's algorithm.

In this new formulation, the optimization problems try to minimize:

$$J = \sum_{k=0}^N \left( \mu_r \left\| \frac{d^4 \mathbf{r}_k}{dt^4} \right\| + \mu_\psi \left\| \frac{d^2 \boldsymbol{\psi}_k}{dt^2} \right\| \right) \quad (5.27)$$

that is minimizing the snap (because the thrust and pitch and roll body rates are expressed as functions of the forth derivative of the position  $\mathbf{r}$ ), and the yaw angular acceleration for minimizing the input relative (in the paper [35] there are all the calculations to express the inputs as function of the snap).

Since this problem does not have a close form, it is written as a quadratic program:

$$\begin{aligned} \min \quad & \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{h}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A} \mathbf{x} \leq \mathbf{b} \end{aligned} \quad (5.28)$$

and then solved with optimization algorithms, finding numerically the result.

This problem requires more time to be solved ( $20ms$ ) w.r.t. the rapid trajectory, so it is not possible to generate multiple trajectories and select the best one at each control loop.

What we are doing is calculating this trajectory taking, among all the final conditions that the state machine is given as input to this module, the one with longest time  $T$  and so producing the trajectory of  $T_{max}$  seconds from the initial state to this point.

### 5.3 Problems with the trajectory generation

With this trajectory generator there are some issues that must be resolved. Right now we have found temporary solutions that can fix these problems but a more proper and robust answer must be found.

Following we report the main problems of this module:

#### 5.3.1 Last chance solution

If both rapid trajectory and minimum snap trajectory do not find a solution we apply this final method.

This method is using the rapid trajectory algorithm calculated with a final time  $T_{feasible}$  such that the trajectory is surely feasible. In the paper [16] there is a method to calculate  $T_{feasible}$  for a trajectory from rest to rest states (initial e final velocity and accelerations equal to 0).

In our case the initial state can also be not static, so the solution should not hold in our case. What we can do is to enlarge the time  $T_{feasible}$ , from rest to rest, by a factor  $\alpha$  proportional to the initial acceleration and velocity, and using this final time to calculate the rapid trajectory.

The minimum time  $T_{feasible}$  is define like the maximum between three different final times. The 3 times are calculate to guarantee the maxim thrust feasibility  $T_{c_{max}}$ , the minimum thrust feasibility  $T_{c_{min}}$  and the body rates feasibility  $T_{\omega_{max}}$ .

We take

$$T_{feasible} = \alpha \max(T_{c_{max}}, T_{c_{min}}, T_{\omega_{max}}) \quad (5.29)$$

because if we calculate a feasible trajectory  $trj_1$ , from static to static states, with terminal time  $T_1$ , and then we calculate  $trj_2$  with  $T_2 \geq T_1$  as new terminal time, then the second trajectory will be surely feasible (this is not always true if initial and final conditions are not resting).

The parameters necessary to find these times are: the distance  $d$  between initial and final position,  $c_{min}$ ,  $c_{max}$  the minimum and maximum thrust, and  $\omega_{max}$  the maximum body rates.

Substituting these variables into the general solution of the optimal control problem, calculating the maximum acceleration that the final trajectory will have, and using 5.19 and 5.20, we can find the three values of time:

$$\begin{aligned} T_{c_{max}} &= \sqrt{\frac{10d}{\sqrt{3}(g - c_{min})}} \\ T_{c_{min}} &= \sqrt{\frac{10d}{\sqrt{3}(c_{max} - g)}} \\ T_{\omega_{max}} &= \sqrt[3]{\frac{60d}{\omega_{max}c_{min}}} \end{aligned} \tag{5.30}$$

At this point  $T_{feasible}$  is defined and we can calculate the rapid trajectory relative.

This trajectory is considered a temporal solution, so as soon as a new initial or final condition arrive, we try to substitute it with a new right trajectory.

Note that the solution found with this last method could not lead to the correct completion of the task in the last two parts of the state machine. In these phases the quadrotor must be in a precise amount of time  $T$  in a specific position, in order to intersect the moving platform, while we are now considering a trajectory with duration  $T_{feasible} \neq T$ . This is why when we use this type of trajectory we try to change it as soon as possible, hoping that at the next initial condition one of the previous two methods do not fail.

### 5.3.2 Too short final time

The rapid trajectory algorithm has another issue: when the final time  $T$  is too short, all the calculated trajectories result indeterminable. This problem is due to the method used to check the feasibility with respect to the input, 5.1.5, but to overcome this problem we can reduce the threshold for which the algorithm stop to recursively control if a piece of the trajectory is feasible.

This way the generation of the trajectory is slower, but we are able to find feasible trajectories when their duration is short.

# Chapter 6

# Experiments

During this thesis, several experiments were performed in order to evaluate the performance of the framework. We did different tests of all the parts of the modules, both in simulation and in the real world, in order to understand the weakness of each module and achieve better results.

In this chapter we describe the hardware used in the real world experiments and the software use for the simulations. Finally, we report the main results achieved during these experiments.

## 6.1 Real world hardware

In all the real world experiments we use only onboard sensing and computing. An OptiTrack motion-capture system [36] is used to have a ground truth values during some experiments, but the quadrotor never uses these data to fly.

### 6.1.1 Quadrotor

We utilize custom-designed quadrotors that are based on 3D printed and electronic parts designed in the RPG lab, combined with some commercial components (see Fig. 6.1).

These platforms are lightweight (around 500 g) and safe for operation in proximity with humans. However, they are also agile while maintaining maneuverability and robust vision-based control: they can achieve a maximum speed of at least  $4 \text{ m s}^{-1}$  during vision-based flight. The quadrotor used is equipped with:

- an IMU that provides linear accelerations and angular rates;
- a quad-core single board computer (Odroid XU4), where all computations described in the previous chapters are performed;
- two different cameras:

- a downward looking camera with a FoV of  $90^\circ$ , used to detect and track the moving platform;
- a forward looking camera with fish-eye lens, used for state estimation. The fish-eye is used to have a wide field of view and be able to track enough features in every configuration.

### 6.1.2 Moving platform

During the experiments we use Jackal UGV as moving platform. Jackal is a small field robotics research platform produced by Clearpath Robotics [37]. It has an onboard computer, GPS, IMU and it is fully integrated with ROS. It can reach a maximum speed of  $2 \text{ m s}^{-1}$ , that is perfect for testing with conditions similar to the final challenge.

Over the UGV we have installed a wooden base  $1m \times 1m$  were we attached both the textures in Fig. 1.2 or the one in Fig. 3.7.

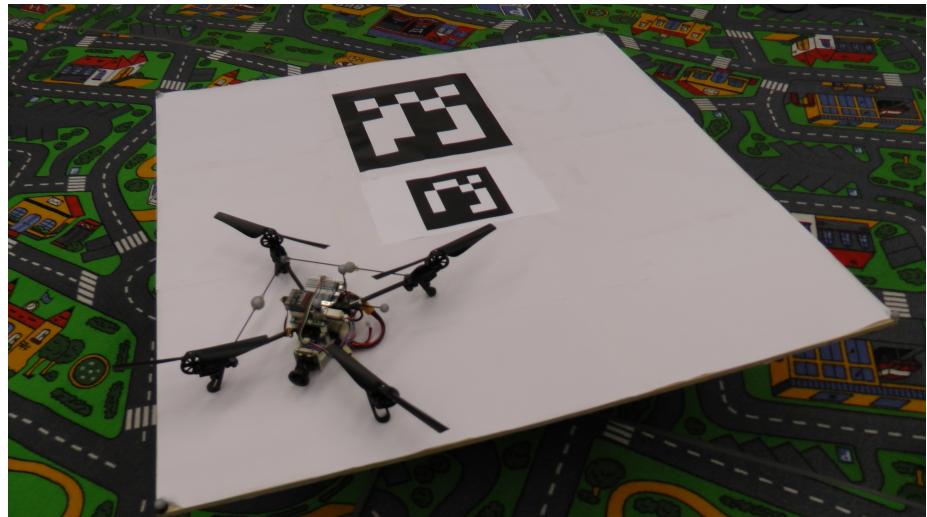


Figure 6.1: The UAV and UGV used during the experiments

## 6.2 Simulation

A simulation environment is developed in order to recreate as precise as possible the final environment of the challenge. As a matter of fact, organizing experiments in real field as large as the one in the challenge, can be difficult, but with the simulation environment we can test the whole framework before trying it in the real world.

The simulation is done using Gazebo simulator [38]: it is a free simulation toolbox useful to reproduce populations of robots in complex indoor and outdoor environments, furthermore this toolbox is directly part of ROS.

To simulate the quadrotor we use the RotorS simulator [39]: a UAV Gazebo

simulator that provides some multirotor models among which there is the As-Tec Hummingbird, very similar to the quadrotor we are using in the real world experiments. All quadrotors can be provided with many sensors such as IMU, cameras etc.

To simulate the moving platform we use the ROS package [40] that allows us to control a Clearpath Husky. Over the UGV we installed a platform identical to the one used in the real world.

The framework used during the tests in the simulation are the same ones explained in Chap. 2, with the difference that the state estimation is not coming from SVO+MSF, but is given by Gazebo and we corrupted it with a Gaussian noise with 0 mean and  $\sigma^2$  variance.

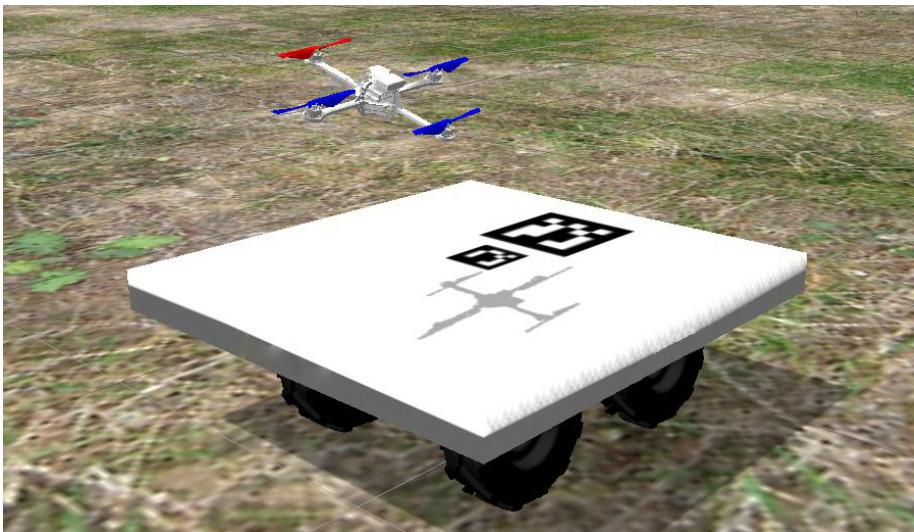


Figure 6.2: The UAV and UGV used during the experiments in simulation

### 6.3 SVO

For this project we need to use a front looking camera for the visual odometry, because, when the UAV is over the base, the majority of the image from the down looking camera is occluded by the platform itself.

This is a crucial problem because the features on the base cannot be considered: the perceived movement is relative to the moving platform and not to the world frame. For example, in the scenario in which the quadrotor and the platform are moving with the same velocity, the images taken from the camera are not changing over time, even if the camera is moving. In this case the visual odometry fails to provide reliable results.

In the final parts of the mission, using the down looking camera, there will be not enough good features to track for UAV state estimation, so is necessary to use a front looking camera for self state estimation.

### 6.3.1 Front looking vs down looking

To compare the results of front looking and down looking SVO we took datasets in which we run two instances of SVO (using the two different images from the two cameras) to compute the pose estimation of the quadrotor and then filter these pose with MSF using the same IMU signal. These state estimations are compared also with the ground truth from the OptiTrack. This way we can compare the two versions of SVO+MSF with the real pose.

The following images show the the results of one of these experiments. In this particular test we hold the quadrotor by hand and we move it inside the flyingroom simulating a square trajectory Fig. 6.3.

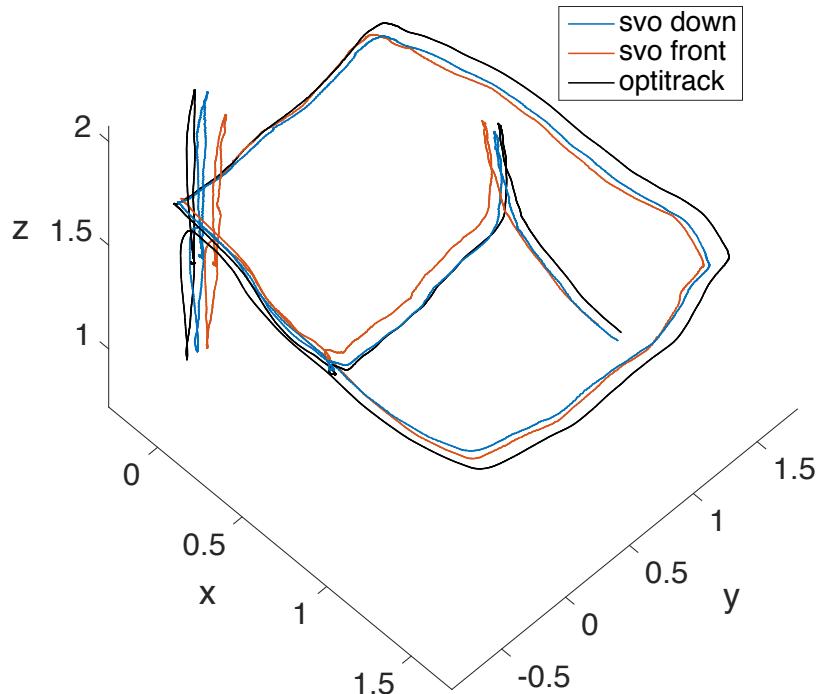


Figure 6.3: Comparison between SVO position estimation in 3D world. The blue line is the estimation with down looking camera, the red line with front looking one and the black line the ground truth given by the OptiTrack

In particular, Fig. 6.4 shows the position , Fig. 6.5 the orientation and Fig. 6.6 the velocity estimations from the two versions of SVO compared to the OptiTrack.

We can see that both versions of our estimation framework provide a reliable estimation of the quadrotor pose and velocity. In order to evaluate the precision of the two estimations, see Fig. 6.7, where the average position error between the two versions of SVO is shown: error generally below 10 cm with RMSE of 4.5 cm for the down looking camera and 6 cm for the front looking one.

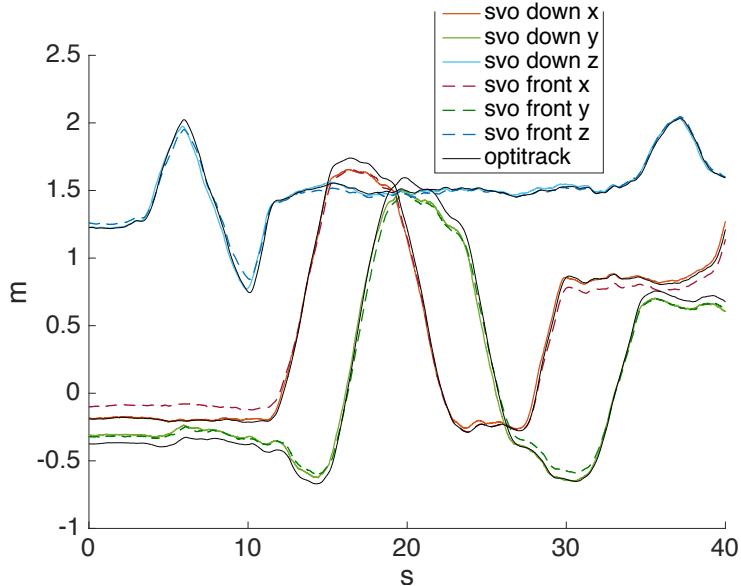


Figure 6.4: Comparison between SVO position estimations with the front looking camera (dashed lines), down looking camera (solid) and optitrack (black).

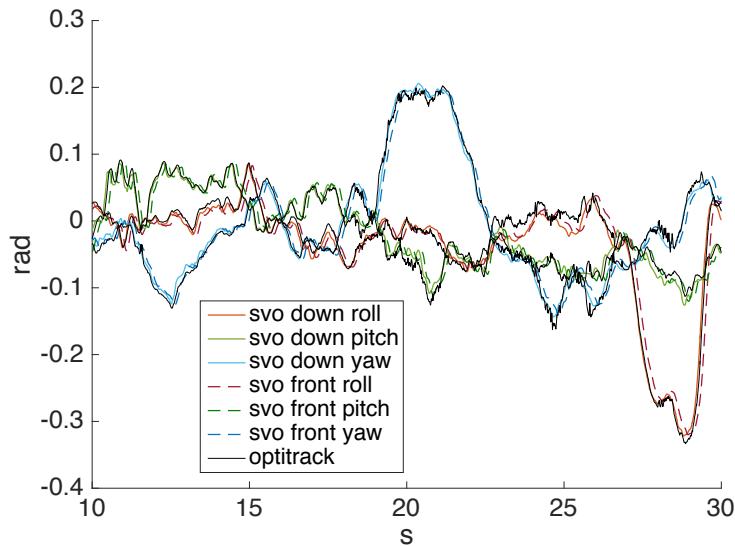


Figure 6.5: Comparison between SVO orientation estimations with the front looking camera (dashed lines), down looking camera (solid) , and OptiTrack (black lines). The orientation data from the OptiTrack are low pass filtered in order to eliminate the high frequency noise.

From Fig. 6.3 we can see that the main problem is the scale factor. The scale factor is estimated at the beginning, tracking features of the images and setting their depth at a known value. If there is an error in this initialization then all the world is scaled and will result bigger or smaller than the reality.

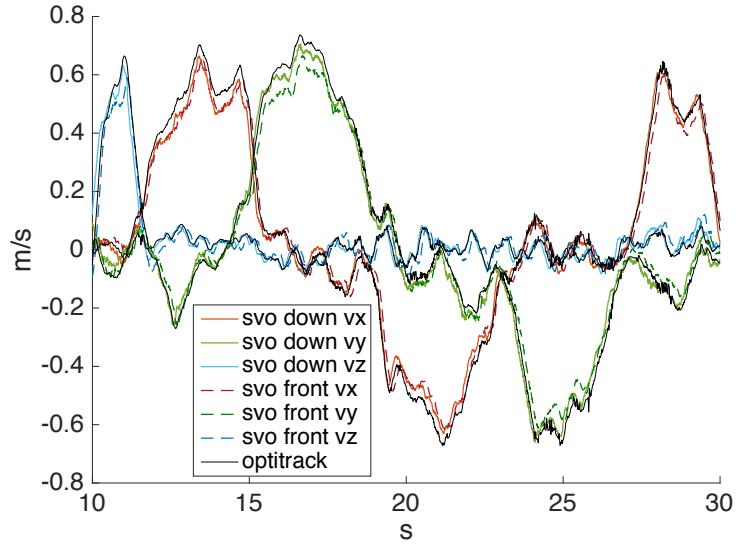


Figure 6.6: Comparison between SVO position estimations with the front looking camera (dashed lines), down looking camera (solid lines), and OptiTrack (black lines). The velocity data from the OptiTrack are low pass filtered in order to eliminate the high frequency noise.

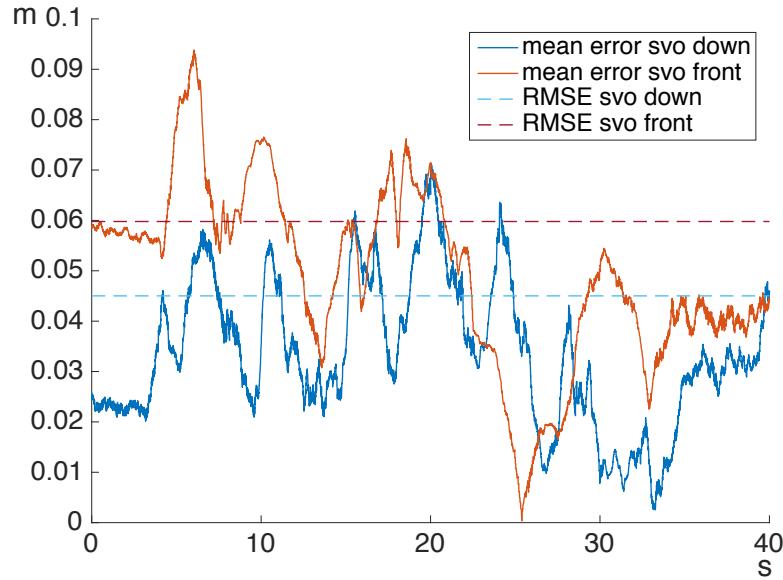


Figure 6.7: Mean error between the 3D position estimations of the two versions of SVO and the ground truth. Blue line is the error with the down looking camera, the red line with the front looking one. The correspondent dashed lines are the RMSE for the two estimations.

## 6.4 Base detection and tracking

Several experiments were performed both in simulation and in the real world to evaluate the performance of the platform state estimation.

### 6.4.1 From high altitude

We do not require the state estimation from high altitude to be very precise, since we need a rough estimation of the position of the platform. The precision of the estimation depends on the altitude from which the quadrotor is tracking the base and, with the designed EKF, we obtain a reliable estimate of the base state.

Figg. 6.8, and show the result of different experiments both in the real world and simulation.

In real world experiments we tested the detector with the platform described before. To have a comparison with a ground truth we did the experiments in the RPG flyingroom, where the position of the platform from the OptiTrack is available. In these experiments the quadrotor cannot reach high altitude and it tracks the moving platform from 2 m.

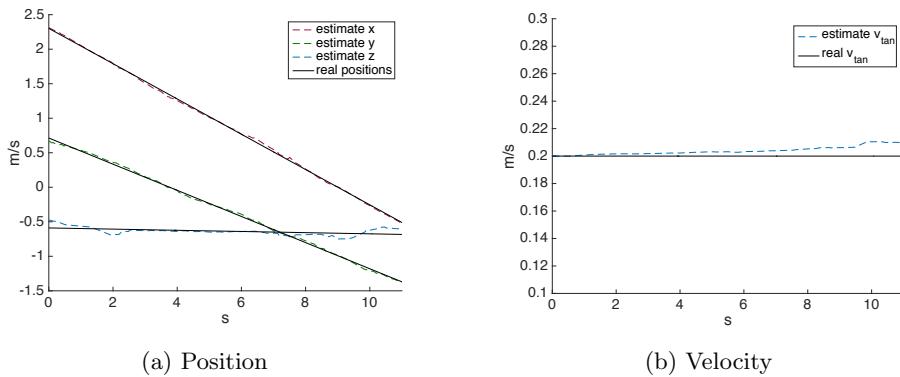


Figure 6.8: Real world experiment. Comparison between estimate position and velocity with the ground truth values for a platform moving at  $0.2 \text{ m s}^{-1}$ . The estimate position is taken from 2 m height and has a RMSE of 5 cm in  $x, y$  and 10 cm in  $z$ .

In simulation we can track the platform from really high altitude. Figure 6.9 shows the result of a tracking from 15 m height. Of course, in this case the estimate is no longer very precise, because the precision of the pose estimation of the platform decreases at high altitude.

As one can see from Fig. 6.10, the estimation error can be really high during this phase, as shown by the average error in  $x$  and  $y$  directions and the correspondent RMSE from a searching at 15 m of altitude. In general, the state estimate has a RMSE of 0.4 m.

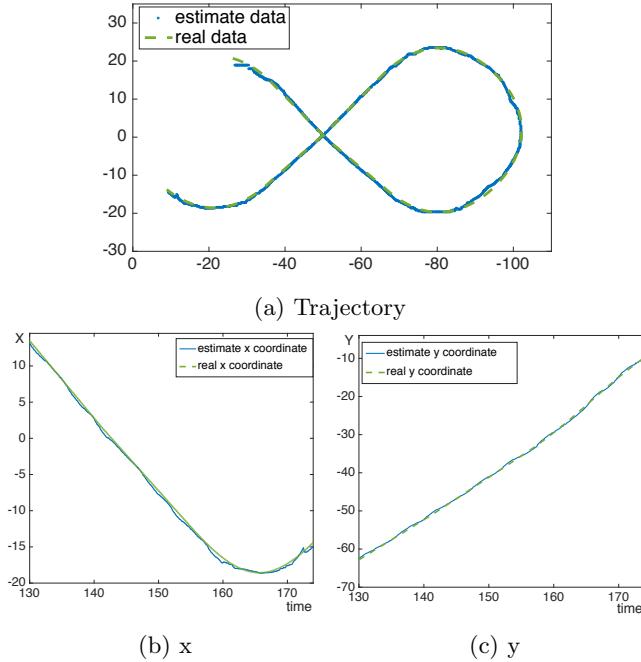


Figure 6.9: Comparison between estimate position (blue dots) and real position (green line) in simulation. The platform is moving in the 8 shape path at  $1.5 \text{ m s}^{-1}$  and the quadrotor explores the area at 15 m of altitude.

Even if they are not very accurate, this data are good enough to perform the first stages of the state machine.

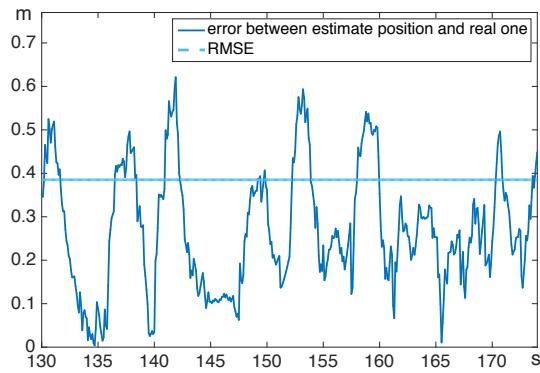


Figure 6.10: Average error between estimate and real x,y coordinate. The RMSE is below 0.4 m. This precision is sufficient to estimate the type of movement of the base.

### 6.4.2 From low altitude

#### Different AR-Tag detector

In the real world implementation we tried several different tag detector ROS packages, such as RPG-April-Tags [41] that uses the AprilTags library [42], AR-Sys [43] and AR-Track-Alvar [44].

All of them have some strengths and weaknesses and we compared the most important features to understand which detector is the more suitable for our purpose:

- **Light conditions:** all these methods use the edge based approach, so the results is similar in different light conditions.
- **Final pose:** all trackers solve a PnP Problem to find the 6DoF pose of the camera that minimizes the reprojection error of the points in the image. The final result is the transformation between the tag and the camera. RPG-AprilTag has also the possibility to return a 4DoF pose (perfect for our application), saving some computation.
- **Multiple tags:** AR-Sys and AR-Track-Alvar have the ability to directly track multiple tags or single target composed by multiple tags.
- **Precision:** we measure the error at 1 m distance from the tag
  - RPG-April-Tags:  $\pm 1$  pixel
  - AR-Sys:  $\pm 2$  pixel
  - AR-Track-Alvar:  $\pm 1$  pixel
- **Frequency:** on the quadrotor the performance of the three tracker where quite different
  - RPG-April-Tags: 1 Hz
  - AR-Sys: 4 Hz
  - AR-Track-Alvar: 1 Hz

**AR-Sys** In our final implementation we decided to use AR-Sys because of its computational efficiency. AR-Sys is 3D pose estimation ROS package that uses ARUco marker boards [45].

This package guarantees a good error correction in the identification of a specific tag, and, more interesting for our application, a solution to the occlusion problem using multiple markers.

It can identify the pose of boards composed by multiple tags considered as a single unit. The board is defined in an XML file where all the tag are listed with an ID and the relative position w.r.t. the master tag (the first in the list) that defines the center of the cumulative target. The pose of the camera is given with respect to this tag. This feature guarantees more stable pose estimates and robustness to the occlusion of a part of the platform.

Very accurate pose estimation is obtain when the AR tags are used. Generally, the error in the  $x, y$  coordinate is less then 7 cm, in the  $z$  direction is about 3 cm.

The following figures show different experiments in the real world and in the simulation with different velocities and initial values.

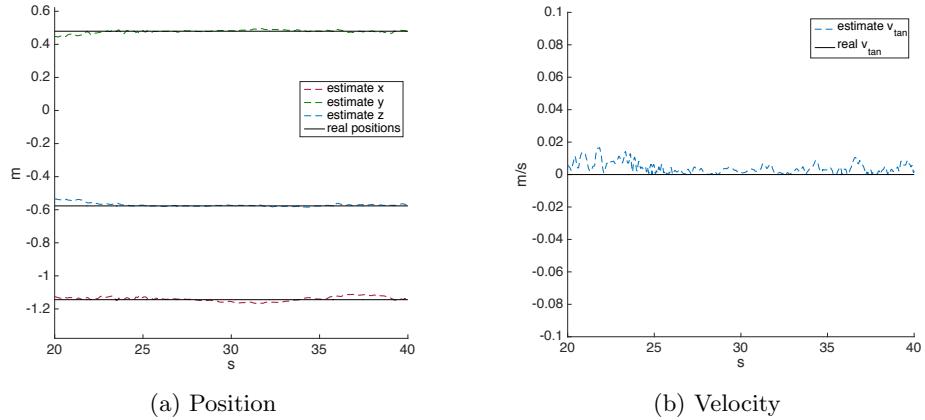


Figure 6.11: Real world experiment. Comparison between estimate position and velocity with the ground truth values for a static platform. The estimate position has a RMSE of 5 cm in  $x, y$  and 2.5 cm in  $z$ .

## 6.5 Trajectory generation

### 6.5.1 Acceleration estimation

We described in Sec. 5.1.6 the three possible methods to estimate the acceleration of the quadrotor at a given time. This calculation is necessary because the trajectory generator needs a full state description of the initial condition of the quadrotor [position,velocity,acceleration] in order to solve the optimal control problem.

We took several datasets while the quadrotor was flying in the flyingroom in order to compare the different methods used to estimate the accelerations.

Figure 6.14 compares the data taken directly from the IMU against finite difference calculated with two subsequent velocity estimations (without filtering) and the estimation computed with the total thrust. On the right column we provide the same comparison, but this time we consider the filtered versions of the accelerations calculated with finite difference and with the data from the IMU.

It is easy to notice that only the raw data from the thrust can be considered a good approximation of the acceleration, while the other two signals need some filtering.

Filtered data from IMU and finite differences are smoother, but filtering the high frequencies we slow down the response of these signal and therefore cause

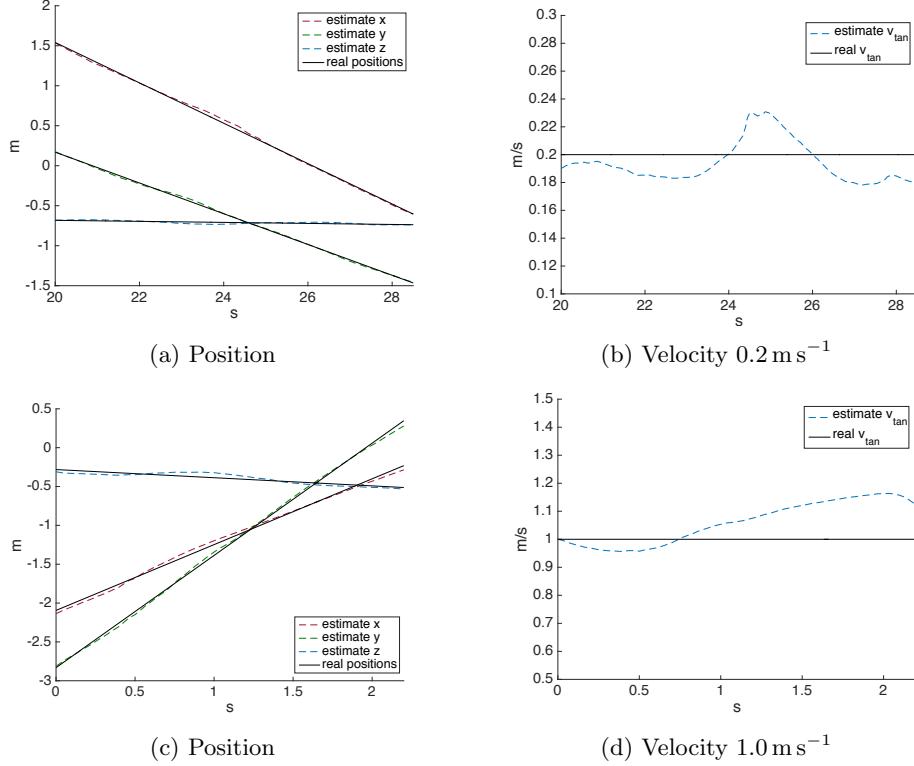


Figure 6.12: Real world experiment. Comparison between estimate position and velocity with the ground truth values for a platform moving at: figures (a)(b)  $0.2 \text{ m s}^{-1}$ , figure (c)(d)  $1.0 \text{ m s}^{-1}$ . The estimate position has a RMSE of 3 cm in  $x, y, z$ .

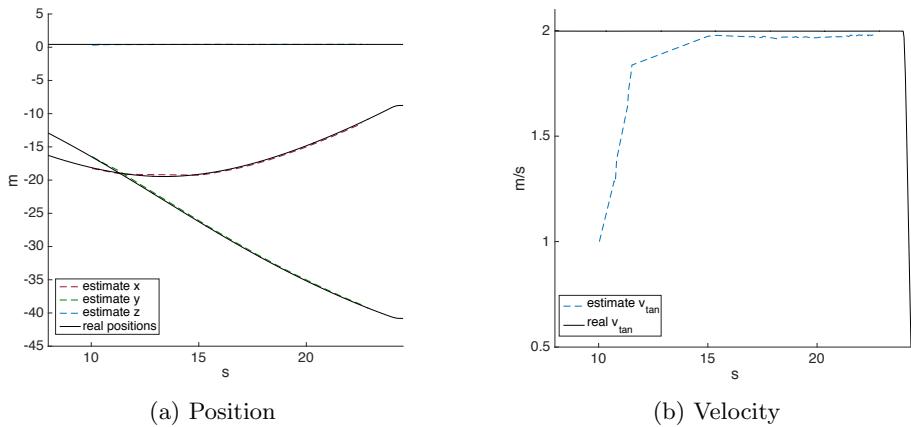


Figure 6.13: Simulation test. Comparison between estimate position and velocity with the ground truth values. The velocity is initialized with a wrong value of  $1 \text{ m s}^{-1}$  but the filter needs few steps to converge to the right value of  $2 \text{ m s}^{-1}$ . The estimate position has a RMSE of 10 cm in  $x, y$  and 2 cm in  $z$ .

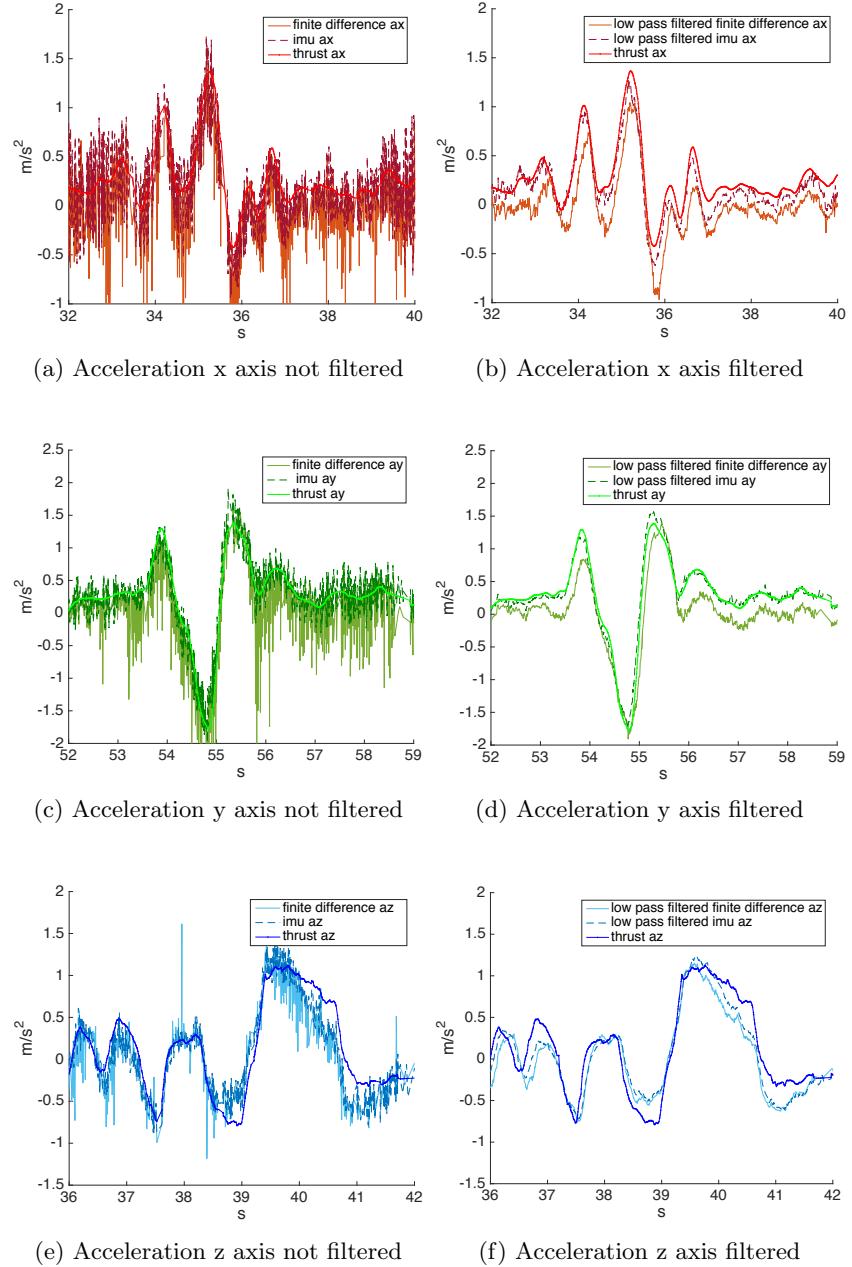


Figure 6.14: Comparison between the accelerations calculate with the three different methods. On the left side the raw data without filtering IMU and finite difference measurements. On the right column the same dataset, but with filtered version for IMU and finite difference.

a delay.

Figure 6.14 shows that there is some offset between the three different approximations: in particular, in the z direction this difference between the acceleration computed with the thrust and the other two methods is very clear.

This offset is more evident in the Fig. 6.15 where the right graph is the same figure 6.14f, while the left part is calculate with the same data set, but the mass of the quadrotor is modify by the 5% from 515 g to 545 g. This tiny modification is creating a big difference in the final acceleration.

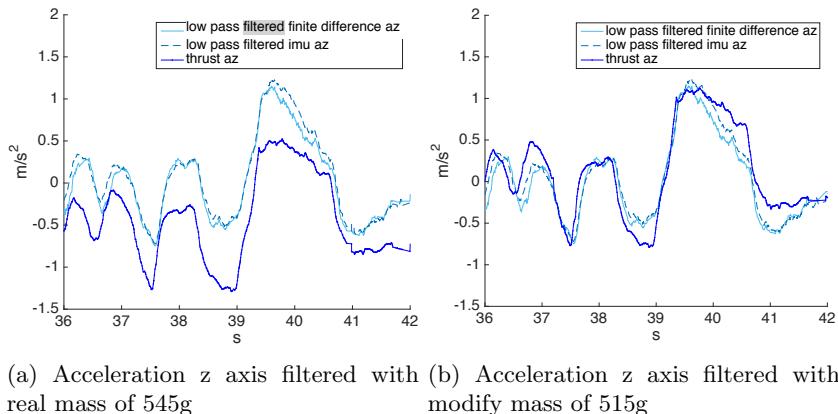


Figure 6.15: Comparison between the accelerations calculate with the three different methods. On the left side accelerations calculate considering the real mass of 545g , on the right column considering 545g

Also in the other axis, if we change the mass, we can see that the acceleration estimation computed with the thrust shows an offset with respect to the other two signals.

This could be caused by a not precise estimation of the rotor fitness factors.

## 6.6 Landing on a moving platform

We also made different trials of the entire framework, in order to understand if all the pieces linked together are good enough to complete the task.

In simulation we reached velocity of the moving platform of up to  $3 \text{ m s}^{-1}$ , with landing rate of over 90%, while in the real world we managed to achieve velocity of  $0.6 \text{ m s}^{-1}$  for the platform. The main issues with higher velocity in the real world is that the state estimation drifts, and the quadrotor we are using has hardware limitations that do not allow fast trajectory tracking, so in the final stages of the state machine, the trajectory generator is not able to find feasible trajectories that bring the quadrotor on the platform.

## Chapter 7

# Discussion

This thesis proposes a landing algorithm for a quadrotor on a moving platform. The approach presented is combining the robustness of predicting and replanning methods with the possibility of onboard computation.

We proved functionality of this framework both in simulation and in real word experiments achieving good results in terms of robustness.

The complete onboard computation ensure the framework to be robust to delays and lost of connection with other equipment, very likely in a large environment as the one in MBZIRC.

On the other end the replanning in an MPC style seems very promising to correct estimation errors, using noisy sensors and flying at fast velocity, it happens very easily.

When the initial condition of the quadrotor are far from the hovering state, the trajectory generator method appears not really robust and it is failing to find a feasible path to complete the task.

### 7.1 Conclusion

In this thesis we presented a complete framework to permits a quadrotor to find, approach and land on a moving platform. We explained all the modules that make up the system, showing in detail the computation we perform in order to complete the assigned task.

A set of algorithms were developed for detecting a tracking the target based on images. The observation and the non-holonomic models were concatenated to formulate a nonlinear estimator (EKF).

A self state estimation based on visual odometry and a controller were integrate and implemented into a single system that achieve the final mission.

Several experiments were carried in simulation and in the real world to demonstrate the functionality of our system. From these experiments we proved robustness of the framework up to a certain velocity, after which the trajectory generation and the self state estimation modules start to fail.

The flight test showed satisfactory performance, but in general we can say that

further work needs to be done in order to achieve results that can be used in the MBZIRC challenge.

## 7.2 Future Work

There are several upgrades that can be done to this frameworks. The major problems are related to the not always robust state estimation and the issues with the trajectory generator ( described in 7.2.3).

Following we describe what solution can be applied in future to solve these problems.

### 7.2.1 SVO with multiple cameras

Since we have more then one camera on the UAV we can use the images from all of them to estimate the pose of the quadrotor with SVO, in this way we have a more robust and accurate state estimation. Furthermore, using more then one camera, we have low probability that all the visual odometry fail in the same time and so we can recover automatically when

### 7.2.2 State estimation fuse multiple sensors

A future upgrade that we should do is to fuse multiple sensor to have a more robust and precise state estimation.

As a matter of fact MSF can combine easily different sources of data, filtering them with the IMU information. The main two sensors we can add for this upgrade can be:

- GPS: it gives a 3D absolute position with not a high accuracy, but are always available in outside environment, and can be useful to have a continuous state estimation used to initialized (and reinitialized if it fails) the visual odometry. Of course the uncertainty related to this measure will be much more higher w.r.t the one from SVO, but it is MSF's duty taking in account these information and filtering the data in the right way.
- Teraranger [46]: it is distance sensor for robotics, it can operate both in inside and outside environment, it is very light and can be really useful to have an estimation of the height of the quadrotor. It is in fact well known that both VO and GPS systems have much more error in the depth component, so the data from this sensor can be correct all the wrong estimations from the other two sources.

### 7.2.3 Change the controller

As describe before the trajectory generator has some issues that must be resolved . The approach to solve these problems can be:

- make the flight controller more sensitive: right now the replanning does not work because the first desired state of the trajectory is too close to the current state to generate a correct control action. Making the controller more responsive at little variation can solve this problem.

A method to increase the sensitivity is tuning the controller gains, but this can lead to an unstable behavior, so further studies must be done.

- change both the trajectory and the controller: implementing a new controller like a LQR controller [47] that takes into account both the dynamics of the quad and the platform and directly calculate the control actions necessary to arrive at a certain final state.

In this case the state machine should not predict in advance where the platform will be in  $T$  seconds because this prediction is directly done by the LQR controller. The main problem with this type of controller is tuning the weights in the cost function to have a nice and smooth flight.

Furthermore we can implement a continuous replanning of the control actions of the LQR, leading with an MPC framework. This solution, as described in the introduction of the thesis, is really computational expensive and before implementing it, we must understand if it can run onboard on our quadrotor.

### 7.2.4 Cross detector

In the final challenge the moving platform will be signed with the marker in figure 1.2.

In order to have a measurement update in the low-altitude EKF we have to implement a cross detector: instead of estimating the 4dof pose of the platform from the AR-tag detection, we must be able to extract the same information from the cross mark.

The detector itself should not be really hard to implement (it consists in a new PnP problem) the only problem can be due to the symmetry of the cross that does not allow to detect a unique solution for the yaw orientation. On the other end once we are estimating the initial yaw angle we are able to detect correctly the changing in orientation, as far as between two consecutive measures the platform rotates a few degrees: we cannot distinguish between rotations of  $k90^\circ$ , but we know that two measures close in time has also close degree because the angular velocity of the platform is not really high.

From the point of view of our framework we can simply substitute the detection module and everything still working: this new detector should provide the same data of the AR-tag detector used in this thesis, and so it can directly be used as update step on the EKF already implemented.

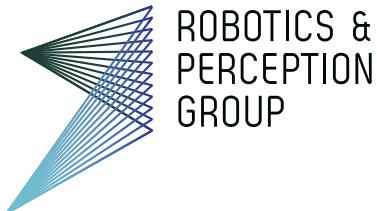
# Bibliography

- [1] Srikanth Saripalli, James F Montgomery, and Gaurav S Sukhatme. Vision-based autonomous landing of an unmanned aerial vehicle. 3:2799–2804, 2002.
- [2] Courtney S Sharp, Omid Shakernia, and S Shankar Sastry. A vision system for landing an unmanned aerial vehicle. 2:1720–1727, 2001.
- [3] Sven Lange, Niko Sünderhauf, and Peter Protzel. Autonomous landing for a multirotor uav using vision. pages 482–491, 2008.
- [4] Bruno Herisse, Francois-Xavier Russotto, Tarek Hamel, and Robert Mähny. Hovering flight and vertical landing control of a vtol unmanned aerial vehicle using optical flow. pages 801–806, 2008.
- [5] Daquan Tang, Fei Li, Ning Shen, and Shaojun Guo. Uav attitude and position estimation for vision-based landing. 9:4446–4450, 2011.
- [6] Zhou Jian, Wang Xin-Min, and Wang Xiao-Yan. Automatic landing control of uav based on optical guidance. pages 152–155, 2012.
- [7] Daniel Mellinger, Michael Shomin, and Vijay Kumar. Control of quadrotors for robust perching and landing. pages 205–225, 2010.
- [8] Karl Engelbert Wenzel, Andreas Masselli, and Andreas Zell. Automatic take off, tracking and landing of a miniature uav on a moving carrier vehicle. *Journal of intelligent & robotic systems*, 61(1-4):221–238, 2011.
- [9] Daewon Lee, Tyler Ryan, and H Jin Kim. Autonomous landing of a vtol uav on a moving platform using image-based visual servoing. pages 971–976, 2012.
- [10] JeongWoon Kim, YeonDeuk Jung, DaSol Lee, and David Hyunchul Shim. Landing control on a mobile platform for multi-copters using an omnidirectional image sensor. *Journal of Intelligent & Robotic Systems*, pages 1–13, 2016.
- [11] Panagiotis Vlantis, Panos Marantos, Charalampos P Bechlioulis, and Kostas J Kyriakopoulos. Quadrotor landing on an inclined platform of a moving ground vehicle. pages 2202–2207, 2015.
- [12] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

- [13] Michael Neunert, Cédric de Crousaz, Fadri Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, and Jonas Buchli. Fast nonlinear model predictive control for unified trajectory optimization and tracking.
- [14] Moses Bangura and Robert Mahony. Real-time model predictive control for quadrotors. *IFAC Proceedings Volumes*, 47(3):11773–11780, 2014.
- [15] Mark W Mueller and Raffaello D’Andrea. A model predictive controller for quadrocopter state interception. pages 1383–1389, 2013.
- [16] Mark W Mueller, Markus Hehn, and Raffaello D’Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics*, 31(6):1294–1310, 2015.
- [17] MBZIRC. Mbzirc challenge description, 2016.
- [18] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. 2009.
- [19] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. pages 15–22, 2014.
- [20] Simon Lynen, Markus W Achtelik, Stephan Weiss, Margarita Chli, and Roland Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. pages 3923–3929, 2013.
- [21] Bradley M Bell and Frederick W Cathey. The iterated kalman filter update as a gauss-newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297, 1993.
- [22] Matthias Faessler, Flavio Fontana, Christian Forster, and Davide Scaramuzza. Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1722–1729. IEEE, 2015.
- [23] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(Series D):35–45, 1960.
- [24] Wikipedia. Runge-kutta methods, 2016.
- [25] Steven S. Beauchemin and John L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995.
- [26] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 81(1):674–679, 1981.
- [27] Juyang Weng, Paul Cohen, Marc Herniou, et al. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on pattern analysis and machine intelligence*, 14(10):965–980, 1992.
- [28] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *IEEE Transactions on pattern analysis and machine intelligence*, 21(8):774–780, 1999.
- [29] Wikipedia. Artag, 2016.

- [30] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. pages 85–94, 1999.
- [31] Mark Fiala. Designing highly reliable fiducial markers. *IEEE Transactions on Pattern analysis and machine intelligence*, 32(7):1317–1324, 2010.
- [32] G. Bradski. OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [33] Serge Belongie. Rodrigues rotation formula. *MathWorld A Wolfram Web Resource*, 1999.
- [34] Michael J Van Nieuwstadt and Richard M Murray. Real time trajectory generation for differentially flat systems. 1997.
- [35] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. pages 2520–2525, 2011.
- [36] Optitrack motion-caption system.
- [37] Clearpath Robotics. Clearpath robotics, 2009.
- [38] Dave Coleman John Hsu, Nate Koenig. Gazebo simulator, 2002.
- [39] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Robot operating system (ros): The complete reference (volume 1). pages 595–625, 2016.
- [40] Clearpath Robotics. Robots husky ros package.
- [41] RPG. Rpg-april-tags ros package, 2014.
- [42] Jeffrey Boyland Edwin Olson, Michael Kaess David Touretzky, and Hordur Johannson. April-tags library, 2012.
- [43] Salinas Bence Magyar Hamdi Sahloul, Rafael Munoz. Ar-sys ros package, 2014.
- [44] Scott Niekum. Ar-track-alvar ros package, 2012.
- [45] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.
- [46] Tera bee. Teraranger sensor.
- [47] Wikipedia. Linear-quadratic regulator, 2016.





**Title of work:**

Autonomous Quadrotor Landing on a Moving Platform with only Onboard Sensing and Computing

**Thesis type and date:**

Master Thesis, September 2016

**Supervision:**

Davide Falanga

Junje Zhang

Prof. Dr. Davide Scaramuzza

**Student:**

Name:	Alessio Zanchettin
E-mail:	zalessio@student.ethz.ch
Legi-Nr.:	97-906-739

**Statement regarding plagiarism:**

By signing this statement, I affirm that I have read the information notice on plagiarism, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Information notice on plagiarism:

[http://www.lehre.uzh.ch/plagiate/20110314\\_LK\\_Plagiarism.pdf](http://www.lehre.uzh.ch/plagiate/20110314_LK_Plagiarism.pdf)

Zurich, 27. 9. 2016: \_\_\_\_\_