ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ROBOTICS &
PERCEPTION
GROUP

Alessio Zanchettin

Autonomous landing on a moving platform

Master Thesis

Robotics and Perception Group
University of Zurich

Supervision

First Supervisor Davide Falanga
Second Supervisor
Prof. Dr. Davide Scaramuzza

September 2016

Contents

Abstract

In this thesis we study the problem of autonomous landing a quadrotor on a moving platform.
The aerial robot employs a forward looking fisheye camera to perform self state estimation and a down looking camera to detect and observe the landing platform, which is 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 searching for the base, finding and approaching the platform, while maintaining it within the camera's field of view, and finally landing on it, minimizing the errors in position, and velocity.
The frameworks consists in several modules that perform different functions and collaborate together to complete the mission. All the computation are onboard, and so the quadrotor can perform this task fully autonomously.
The system is validated in the real world on a quadrotor: the vehicle successfully landed on the moving platform during outdoor flight tests.

Nomenclature

Notation

$\mathbf{J}$	Jacobian
$\boldsymbol{r}$	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
${}_W\mathbf{t}_{WB}$	translation of B with respect to W, expressed in coordinate system W
$\boldsymbol{w}_{\hat{WB}}$	skew symmetric matrix
$\boldsymbol{c}$	thrust vector with respect to frame B
$\boldsymbol{g}$	gravity with respect to frame W

Scalars are written in lower case letters (a), vectors in lower case bold letters ($\mathbf{a}$) and matrices in upper case bold letters ($\mathbf{A}$).

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 and many applications have been trying to use these vehicles in more and more difficult settings. For many of these scenarios 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, it is highly possible, that a these applications require the landing target to be moving, for example it can be a car during a reconnaissance, so 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, but this sensor can have errors up to 5 meters radius, and landing with such low-quality state estimation will have an almost certain probability of failure. Fortunately, many applications require 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 perform an entire landing task. The main parts of the framework are:

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

1.1 Related Work

During the last 15 years several methods where 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 leads to restrictions in autonomy on the UAV.

At the beginning the research was focused on landing on a static platform.
Hardware and techniques used to achieve the successful completion of the task were various.

Some of them, like Saripalli in [1], presents 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 of this problem.

Other works, like Sharp in [2] and Lange in [3], are using little UAVs with cameras, but they are estimating the pose of the quadrotor only w.r.t the landing base, that consists on a single tag, so these frameworks are not robust to the loss of the tag and to the noise introduced using just a single reference for the state estimation.
Another similar work is done by Herisse in [4], where only optical flow information is used for hovering flight and vertical landing control.
All these framework works correctly but the main difference with our approach is that their frameworks consist just in the landing maneuver and so the final target is always in the f.o.v. of the camera, an assumption that for as is not possible to do.

Other papers present promising theoretical algorithm to perform a smooth and precise landing, but only tested in simulation, like Tang in [5] where it develop a landing framework based on N-points algorithm and orthogonalization to estimate the state of the aircraft, or like Jiang in [6] where he developed a theoretical optical guided landing control system and its corresponding guidance control law.

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

Wenzel in [7] is performing tracking and landing on a moving base with a small quadrotor. All the experiments are in inside environment, because he is using IR camera, sensors not robust to outside conditions, which allows robust tracking of a pattern of IR lights without direct sunlight. It achieve precise and consistent results with a platform moving both in a circular path or emulating a ship turning on water, but the movement is not fast, $0.4m/s$.

Lee in [8] is using visual servoing to perform the landing maneuver: they develop a feedback control based on the position of the target in the camera image, this idea is interesting because we can always assume that the landing platform has some distinctive features to use to identify the final position where the UAV should go in order to properly land.
Also in this case the landing base is moving slow $0.07m/s$ and is moving always

in a straight line. To control the quadrotor to the landing site they are using a Sliding Mode Control, this method can deal with non linearity of the dynamics and external modeled noise (like the model of the ground effect force).

Kim in [9] uses color filter to find the landing target, the platform has a color unique in the environment and this feature can be spoil to find it easily, furthermore he uses an omnidirectional camera to have always the target in the field of view. Given the measurement of the position of the camera he implements an Extended Kalman Filter to filter the noise and predict the future position of the target. When the state of the moving base it is estimate, the knowledge of the type of movement the platform is performing, can be crucial in order to filter noisy measures and predict where the target will be within t seconds. Once the future position is calculate a trajectory, in position and velocity, is computed from the initial pose of the quadrotor to the final intersection point. A velocity-attitude control is implement to follow the trajectory, and the precomputed trajectory is followed until the end, without replanning.

Mellinger in [10] is addressing another type of problem: landing on tilted surface in which the quadrotor must pearch. He uses motion capture system in order to have both UAV and target state estimation. His algorithm consists in a precomputed trajectory followed by a position-altitude control based on the linearized system 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.

Vlantis in [11] studied 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 he developed a discrete-time non-linear Model Predictive Controller [12] that optimizes both the trajectories and the time horizon, while respecting input and state constraints (not collide with the platform).
The cost function of the MPC consists in different factors weighted with dynamic coefficients (function of the relative position between UAV and moving platform). There are classical therms related to the 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. This method seems really effective, but 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: it is necessary a ground station that carries the huge amount of calculation that MPC requires.

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

- a good estimation of the UAV's state and moving platform's state.
- a "manager" that considering the state estimations define a current stage of the system, and assign a specific task for this phase.

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

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 colled Sqeuqntial Linear Quadratic (explained in [14]), generating feedforward and feedback controls actions. This method allows agile flight maneuvers with accurate tracking. All the calculations are made on an on-board i7 CPU achieving a rate slightly below $40Hz$. The major drawback for us is that we have also perform many other computations on the on-board computer, so it is not possible to reach this frequency in the controller.

In the paper [15] Bangura presents a solution to on-board trajectory tracking control of 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 computed onboard.

Mueller in [16] and later in [17] 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 the 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. Million motion primitives may be evaluated and compared per second, and so the best feasible one can be picked and followed.

This final paper is very promising for our purpose because our problem can be exactly be expressed as the one solved by the algorithm proposed, and also because it is computationally inexpensive and so it is possible to run the entire code directly onboard.
Furthermore it is possible to use this code in an MPC-style: we have a controller with rate $\frac{1}{dt}$, at initial time t_0 the entire trajectory (from t_0 to $t_0 + T$) is calculated but only the first desired state of the trajectory (related to time $t_0 + dt$) is considered, then at the next control cycle we repeat the procedure calculating the trajectory from $t_0 + dt$ to $t_0 + T$, and so on.

The following table summarizes the work done in previous research on this topic.

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

1.2 MBZIRC challenge

The main reason for the development of this thesis is the participation in the Mohamed Bin Zayed International Robotics Challenge [18]. MBZIRC is an international robotics competition, held every two years, that provides an ambitious and technologically demanding set of challenges, that aim to inspire the future of robotics.
In the description of the competition they motivate the challenge claiming that robotics have an increasing impact in a variety of new markets and on various human social aspects (for example disaster response, oil and gas, manufacturing, construction and household chores) and MBZIRC should lay the foundations of technologies for such applications include robots working more autonomously in dynamic, unstructured environments, while collaborating and interacting with other robots and humans.

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:

- 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 on the track.

1.2.1 The Arena

The challenge will be performed in an arena with the following characteristics (see figure 1.1):

- Outdoor open arena with GPS access.
- Dimension: approximate $100m \times 60m$.

- Track: width 3m, in the shape of a figure 8 (of infinity shape), with boundary marked with white paint.
- Terrain: relatively smooth and relatively level.
- UAV initial start location: 10m away from the arena.

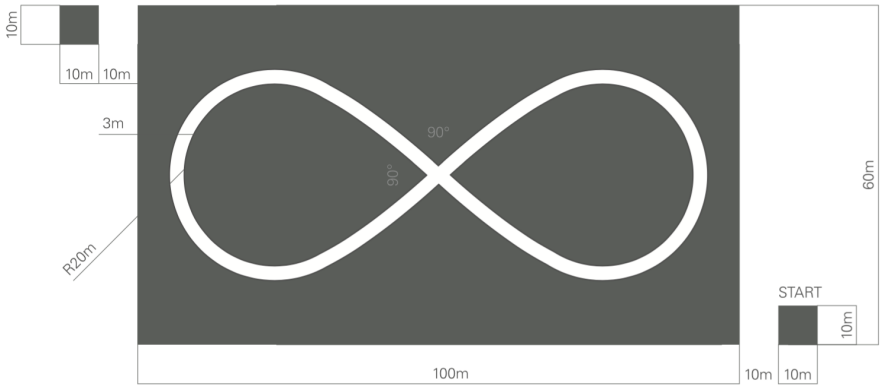


Figure 1.1: Arena of the challenge

1.2.2 Landing platform

The landing platform is mounted on a ground vehicle of approximate dimensions $2.5m \times 1.5m \times 1.5m$ (length, width, height). The moving car starts at a constant speed of $15km/h$, it reduces the speed to $10km/h$ after 6 minutes and to $5Km/h$ after 12 minutes.
The landing platform will be made of a ferrous surface to enable docking using magnetic or suction or other means. It is a square of dimensions $1.5m \times 1.5m$, and approximately $1.5m$ above ground, positioned on the vehicle. The landing zone inside the landing area is a circle of diameter $1m$. The center of the circle is indicated by an X. The landing area, the landing zone and the marker X are shown in Figure 1.2. A landing is considered successful when a point of contact of the UAV is within the landing circle, with propulsion off and rotors not spinning.

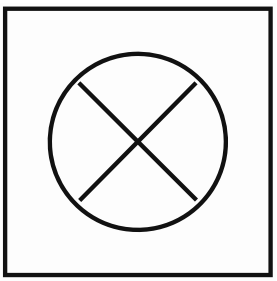


Figure 1.2: Design of the platform in which the quadrotor must land on

Chapter 2

General Framework

Our framework consists in several parts, each one is computing a different subtask and they are communicating one with the other 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 are giving a brief introduction on each part and in the following chapters we will discuss in detail the parts developed in this thesis.

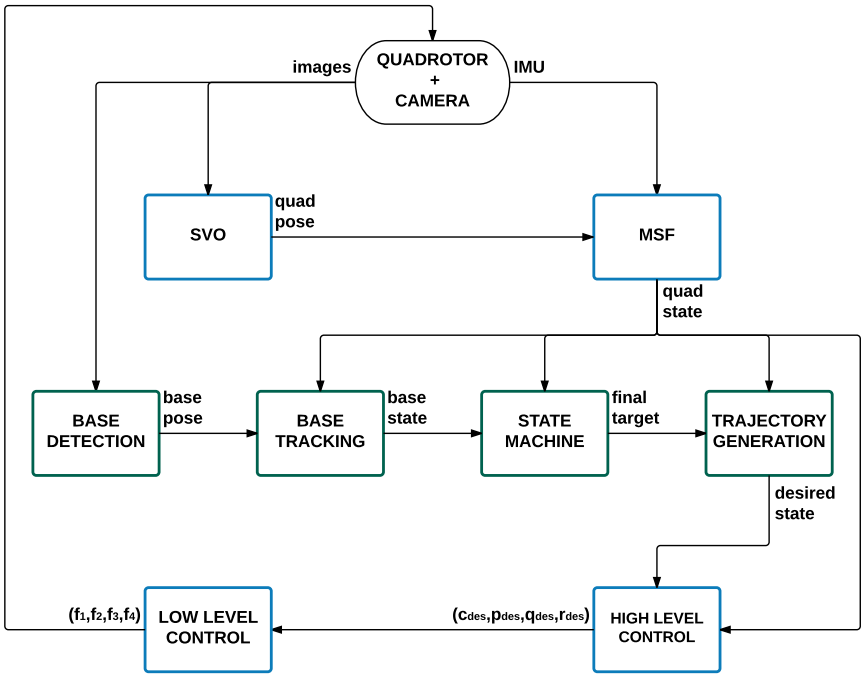


Figure 2.1: Pipeline of the framework. In blue the modules already implemented we add to the work of this thesis. In green the steps developed in this work.

The modular system developed, utilize the Robot Operating System (ROS) [19] for interprocess communication. The platforms used in this project are capable of autonomous, vision-based flight utilizing only their on-board hardware resources, they do not need to rely on external communication infrastructure to accomplish their tasks. In a field robotics scenario with uncertain communication reliability like the MBZIRC, this is of critical importance, 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 provided by the visual odometry algorithm: Semidirect Monocular Visual Odometry (SVO) [20]. It provides a precise and robust pose estimate (3D positiona and orientation), and it runs at up to 55 frames per second on the onboard embedded computer of our flying robots. This vision-based state estimate is fused with inertial data and gravity aligned using the Multi-Sensor Fusion framework (MSF) [21].
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 in the extraction of a sparse set of features in each image, match them in successive frames using invariant feature descriptors, reconstruct camera motion and structure using epipolar geometry and finally, refine 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 insert in 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) [22] 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 given by SVO.
MSF has a modular structure that can support and marge an unlimited number of sensors that give relative or absolute measurements (pose, position, pressure, etc). Furthermore it estimates the calibration states 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 High & low level controls

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 letter that tracks a desired thrust and body rates.

2.2.1 High part

The high level control takes as input:

- $([\hat{x}, \hat{y}, \hat{z}], [\hat{\phi}, \hat{\theta}, \hat{\psi}], [\hat{vx}, \hat{vy}, \hat{vz}], [\hat{p}, \hat{q}, \hat{r}])$: the current state estimate (position, orientation, velocity, angular velocity) , calculate in the previous module
- $([x_{ref}, y_{ref}, z_{ref}], [vx_{ref}, vy_{ref}, vz_{ref}], [ax_{ref}, ay_{ref}, az_{ref}], [\psi_{ref}])$: a reference state (position, velocity, acceleration, yaw) , that can be sampled from a trajectory, calculate in another module, that the UAV should track to perform a task

and gives in output:

- c_{des}: the desired normalize thrust
- $(p_{des}, q_{des}, r_{des})$: 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, synchronized at 50Hz.

Position Controller

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

$$\begin{bmatrix} ax_{des} \\ ay_{des} \\ az_{des} \end{bmatrix} = P_{pos} \begin{bmatrix} x_{ref} - \hat{x} \\ y_{ref} - \hat{y} \\ z_{ref} - \hat{z} \end{bmatrix} + D_{pos} \begin{bmatrix} vx_{ref} - \hat{vx} \\ vy_{ref} - \hat{vy} \\ vz_{ref} - \hat{vz} \end{bmatrix} + \begin{bmatrix} ax_{ref} - \hat{0} \\ ay_{ref} - \hat{0} \\ az_{ref} - \hat{g} \end{bmatrix} \tag{2.1}$$

Where $P_{pos} = diag(p_{xy}, p_{xy}, p_z)$ and $D_{pos} = diag(d_{xy}, d_{xy}, d_z)$ are gain matrices.

Now is very simple to derived the desired normalized thrust c_{des}: it is the projection of a_{des} on the current z axes of the UAV:

$$c_{des} = a_{des} \cdot e_z^B \tag{2.2}$$

Attitude Controller

The output of this controller is a_{des} and together with ψ_{ref} encoded the desired orientation: Since the quadrotor can only accelerate in direction of the z direction of the body we want to align this axis with the desired acceleration a_{des}, so it enforce both ϕ_{des} and θ_{des}. The third degree of freedom is given by ψ_{ref}. With some geometric calculations it easy to define $(p_{des}, q_{des}, r_{des})$: these values are just function of the current orientation $(\hat{\phi}, \hat{\theta}, \hat{\psi})$, the desired final orientation of the z axis $e^B_{z,des} = \frac{a_{des}}{||a_{des}||}$ and the desired final yaw $\psi_{des} = \psi_{ref}$.

TODO calculations??

2.2.2 Low part

The low level control takes as input:

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

and gives in output:

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

We can compute the desired torques $\boldsymbol{\tau}_{des}$ with the feedback linerized scheme:

$$\begin{bmatrix} \tau p_{des} \\ \tau q_{des} \\ \tau r_{des} \end{bmatrix} = JP_{att} \begin{bmatrix} p_{des} - \hat{q} \\ q_{des} - \hat{q} \\ r_{des} - \hat{r} \end{bmatrix} + \begin{bmatrix} \hat{q} \\ \hat{q} \\ \hat{r} \end{bmatrix} \times J \begin{bmatrix} \hat{q} \\ \hat{q} \\ \hat{r} \end{bmatrix} \tag{2.3}$$

Where $P_{att} = diag(p_{qp}, p_{qp}, p_r)$ is a gain matrix and $J = diag(J_{xx}, J_{yy}, J_{zz})$ is the inertia matrix for rotation around the center of mass.
Now to find the thrusts for each rotor we have to solve:

$$\begin{bmatrix} f_{1,des} \\ f_{2,des} \\ f_{3,des} \\ f_{4,des} \end{bmatrix} = \begin{bmatrix} \frac{1}{4\lambda_1}\left(mc_{des} + \frac{\tau r_{des}}{\kappa} - \frac{\sqrt{2}\tau q_{des}}{l} + \frac{\sqrt{2}\tau p_{des}}{l}\right) \\ \frac{1}{4\lambda_2}\left(mc_{des} - \frac{\tau r_{des}}{\kappa} - \frac{\sqrt{2}\tau q_{des}}{l} - \frac{\sqrt{2}\tau p_{des}}{l}\right) \\ \frac{1}{4\lambda_3}\left(mc_{des} + \frac{\tau r_{des}}{\kappa} + \frac{\sqrt{2}\tau q_{des}}{l} - \frac{\sqrt{2}\tau p_{des}}{l}\right) \\ \frac{1}{4\lambda_4}\left(mc_{des} - \frac{\tau r_{des}}{\kappa} + \frac{\sqrt{2}\tau q_{des}}{l} + \frac{\sqrt{2}\tau p_{des}}{l}\right) \end{bmatrix} \tag{2.4}$$

where κ is the rotor-torque coefficient and λ_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 (all of them are parameters of the quadrotor).
Depending on the chosen orientation of the body frame we have a different map between torques τi_{des} and thrusts f_j, in our case the figure 2.2 shows how our coordinate system is oriented w.r.t the 4 propellers.

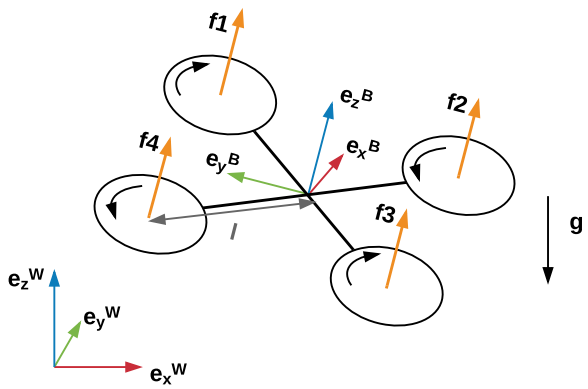


Figure 2.2: Quadrotor with body coordinate frame and thrust forces.

2.3 Base detection & tracking

In order to land on the the moving platform it is necessary to know where the base is at a specific time t. To do so we are using images from the camera to localize where the platform is w.r.t the quadrotor. Then, given the position of the UAV w.r.t the world frame (estimate by the module 2.1), we can reconstruct

the pose of the moving base in the global coordinate frame.
Now if we know how the platform should move in the world frame we can combine the noisy information from the measurements with the theoretical pose that it should have, in order 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 framework. This module implements the manager that decides in which stage the UAV is, based on its pose (estimate by the module 2.1), the position of the base (from the module 2.3), and other inputs.
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 arrive there.

This module will be discussed extensively in chapter 4.

2.5 Trajectory generation

This module of the framework is taking as inputs the quadrotor state estimation (from the module 2.1) and the final state that it has to reach (calculate by the module 2.4) and calculate the trajectory that the UAV must track to reach the final state in T seconds.
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 high-controller module 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 estimation of the odometry of the moving platform. This is necessary in order to have a good prediction of the final state that the quadrotor must have in order to proper 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 vector in world coordinate frame, so 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 produces estimates of unknown variables that are more precise and smooth than those based on measurement alone. This filter is done using Bayesian inference and estimating 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:

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{3.1}$$

- Once the outcome of the next measurement is observed:

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k \tag{3.2}$$

  these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty.

In these equations $\boldsymbol{x}_k$ is the state, $\boldsymbol{z}_k$ the measure, $\boldsymbol{u}_k$ is the control input, f the state dynamics, h the measure model, $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are the process and observation noises which are both assumed to be zero mean multivariate Gaussian noises with covariance $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ respectively.
In the EKF, 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 it can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix, no additional past information is required.
The KF does not require any assumption that the errors are Gaussian. However, the filter yields the exact conditional probability estimate in the special case that all errors are Gaussian distributed.

Following we summarize the mathematical passages we have to perform in order to calculate the final estimation. We consider the case in which the prediction and the update models are discrete in time.

Initialization

$$\begin{aligned} \hat{\boldsymbol{x}}_{k|k-1} &= f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k) \\ \boldsymbol{P}_{k|k-1} &= \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^{\top} + \boldsymbol{Q}_k \end{aligned} \tag{3.3}$$

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

$$\boldsymbol{F}_{k-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k} \tag{3.4}$$

While the update equations yield to the following update step:

$$\begin{aligned} \boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^{\top}(\boldsymbol{H}_k\boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^{\top} + \boldsymbol{R}_k)^{-1} \\ \hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1})) \\ \boldsymbol{P}_{k|k} &= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k|k-1} \end{aligned} \tag{3.5}$$

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

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{3.6}$$

3.1 Prediction update: non-holonomic model

The platform is considered as a car and simulated with a non-holonomic model 3.1. In this model the state is defined as $\boldsymbol{x} = (x, y, z, \theta, v_{tan}, \phi)$: It corresponds to the 3 position in a space (x, y, z) and the yaw angle of the platform (θ) w.r.t. the world frame, the forward velocity (v_{tan}) and the angle of the front wheels (ϕ). 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 $u_2 = \dot{\phi}$.

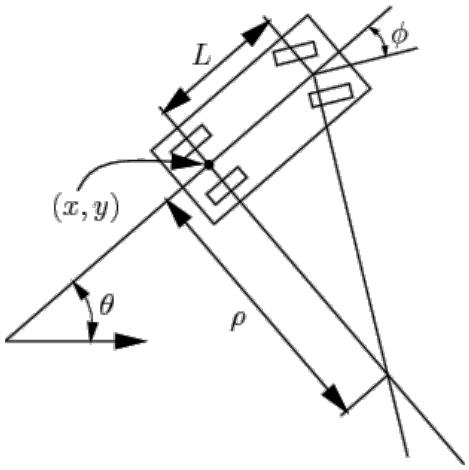


Figure 3.1: Non-holonomic model

The equation of motion in continuous time are:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \\ \dot{v_{tan}} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v_{tan} cos(\theta) \\ v_{tan} sin(\theta) \\ 0 \\ \frac{v_{tan}}{L} tan(\phi) \\ u_1 \\ u_2 \end{bmatrix} \tag{3.7}$$

It is possible to discretize these dynamics in $t \in [t_{k-1}, t_k]$ with a first order finite difference:

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

$$\boldsymbol{x_k} = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_k \\ v_{tan,k} \\ \phi_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + dt\big(v_{tan,k-1} cos(\theta_{k-1})\big) \\ y_{k-1} + dt\big(v_{tan,k-1} sin(\theta_{k-1})\big) \\ z_{k-1} \\ \theta_{k-1} + dt\Big(\frac{v_{k-1}}{L} tan(\phi_{k-1})\Big) \\ v_{tan,k-1} + dt\big(u_{1,k}\big) \\ \phi_{k-1} + dt\big(u_{2,k}\big) \end{bmatrix} \tag{3.8}$$

In order to solve the former system, we have anyway to find a numerical solution. 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:

$$\begin{cases} \dot{\boldsymbol{y}} = f(\boldsymbol{y}, t) \\ \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \end{cases} \tag{3.9}$$

$\boldsymbol{y}$ is an unknown function of time t, which we would like to approximate and the function f and the data t_0, $\boldsymbol{y}_0$ are given.
Now pick a step-size $h > 0$ and define

$$\begin{aligned} \boldsymbol{y}_{k+1} &= \boldsymbol{y}_k + \tfrac{h}{6}\left(\alpha_1 + 2\alpha_2 + 2\alpha_3 + \alpha_4\right), \\ t_{k+1} &= t_k + h \end{aligned} \tag{3.10}$$

for $k = 0, 1, 2, 3, ..., N$ using

$$\begin{aligned} \alpha_1 &= f\Big(\boldsymbol{y}_k, t_k\Big) \\ \alpha_2 &= f\Big(\boldsymbol{y}_k + \frac{h}{2}\alpha_1, t_k + \frac{h}{2}\Big) \\ \alpha_3 &= f\Big(\boldsymbol{y}_k + \frac{h}{2}\alpha_2, t_k + \frac{h}{2}\Big) \\ \alpha_4 &= f\Big(\boldsymbol{y}_k + h\alpha_3, t_k + h\Big) \end{aligned} \tag{3.11}$$

Here $\boldsymbol{y}_{k+1}$ is the RK4 approximation of $\boldsymbol{y}(t_{k+1})$, and it is determined by the present value ($\boldsymbol{y}_k$) plus 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 models, even if they are really simple, can incorporates a large set of possible trajectories.

3.1.2 Infinity shape path

As described in charapter , the moving platform will move in an infinity-shape path described in the figure 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 ρ_8m
- the path is making a cross in the middle that creates 4 angles of $\frac{\pi}{2}$

The easiest way to describe this path is to define how the angle θ is changing in function of the space.
It easy to see that the shape can be seen as a combination of a cross and two circles. The cross is simply defined as the union between the two line, without loss of generality we consider this intersection the origin:

$$\begin{cases} y = x \\ \\ y = -x \end{cases}$$

while the two circles

$$\begin{cases} y^2 + (x - x_0)^2 = \rho_8^2 \\ \\ y^2 + (x + x_0)^2 = \rho_8^2 \end{cases}$$

It easy to see that if we want the intersections between these two functions to be exactly in the 4 points we have to choose

$$x_0 = \frac{\sqrt{2}}{2}\rho_8$$

That correspond to the 4 intersections coordinate

$$\Big(\frac{\sqrt{2}}{2}\rho_8, \frac{\sqrt{2}}{2}\rho_8\Big); \Big(\frac{\sqrt{2}}{2}\rho_8, -\frac{\sqrt{2}}{2}\rho_8\Big); \Big(-\frac{\sqrt{2}}{2}\rho_8, -\frac{\sqrt{2}}{2}\rho_8\Big); \Big(-\frac{\sqrt{2}}{2}\rho_8, \frac{\sqrt{2}}{2}\rho_8\Big)$$

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:

$$\theta(x) = \begin{cases} -\frac{x}{\rho_8} & x \in \Big[0, \frac{3\pi}{4}\rho_8\Big] \\ \\ -\frac{3\pi}{4} & x \in \Big[\frac{3\pi}{4}\rho_8, \frac{3\pi}{4}\rho_8 + \rho_8\Big] \end{cases} \tag{3.12}$$

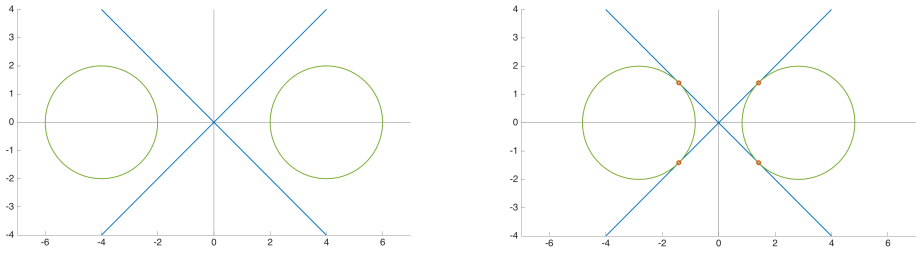


Figure 3.2: How to construct the infinity-shape path

This function define a quarter of the trajectory 3.3b in function of the radius ρ_8 of the path.
It is now possible to use it to generate the entire trajectory $(x(t), y(t))$ 3.3c.
We know that the length of the path is

$$l = 4\Big(\frac{3\pi}{4}\rho_8 + \rho_8\Big)$$

And given the constant velocity v_{tan} we can calculate the time to complete the trajectory

$$T = \frac{l}{v_{tan}}$$

and it is simple to define $\theta(t)$ just stretching or shrinking $\theta(x)$.

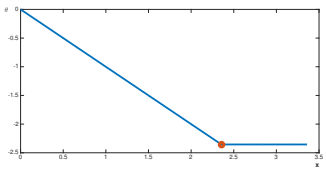
So we can now define:

$$\begin{cases} \dot{x} = v_{tan} cos\big(\theta(t)\big) \\ \dot{y} = v_{tan} sin\big(\theta(t)\big) \end{cases} \tag{3.13}$$

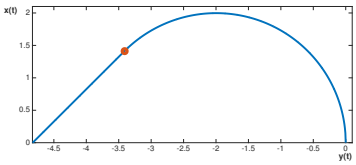
And finally we also need the discretized verison obtain just by forward Euler approximation:

$$\begin{cases} x_k = x_{k-1} + dt\Big(v_{tan,k-1} cos\big(\theta_{k-1}\big)\Big) \\ y_k = y_{k-1} + dt\Big(v_{tan,k-1} sin\big(\theta_{k-1}\big)\Big) \end{cases} \tag{3.14}$$

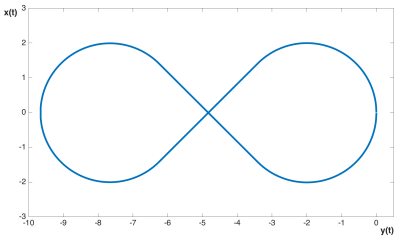
From these calculations we can conclude that the final trajectory of the platform is only a composition of a linear and a circular movement (for a given determinate amount of time). So the model described in 3.1.1 can be used also to express this trajectory.



(a) Angle in function of the space



(b) One quarter of the path



(c) The whole the path

Figure 3.3: The parametrization of the path

3.2 Measurement update

From equation 3.8 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 have the second step of our EKF.
What we are using is a camera with which we are able to identify the moving platform and estimate its relative position and orientation. At this point knowing the position of the camera in the real world we can measure:

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k)$$

$$\boldsymbol{z}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_k \\ v_{tan,k} \\ \phi_k \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_k \end{bmatrix} \tag{3.15}$$

It corresponds to the 3 position in a space (x, y, z) and the yaw angle of the platform θ w.r.t. the world frame. Notice that the observation model h, in this case, is a linear function.

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

- 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 just a few features of the moving car and then the pose estimation are really noisy. Furthermore we do not have any assumption on the initial condition of the platform, we just know the magnitude of constant forward velocity $|v_{tan}|$, so we do not know before if at a certain time t the car is moving in a straight line or in a curve, we have to estimate it, and this is possible only tracking the platform for several seconds.

- 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, starting a more precise measure (base on tag detection), and filtering the measurements with the correct theoretical model of the movement.

3.2.1 From high altitude

To find the car we assume that the platform is the only white square moving on the arena.
Base on this assumption, we analyze the images 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 passages:

- threshold the image in order to find the white features.
- find all the close shapes in the image.
- select only the shapes with
  - 4 edges
  - convex contour
  - angles between edges close to $\frac{\pi}{2}$

At this point we have the position of the four corners of all the squares in the image.
Now we try to 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] try to calculate the motion, for each pixel, between two image frames which are taken at times t and $t + \Delta t$.
For a 2D dimensional case a pixel at location (x, y, t) with intensity $I(x, y, t)$ will have moved by Δx,Δy and Δt between the two image frames.
To solve this problem, the core assumption is the brightness constancy constraint:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{3.16}$$

Assuming the movement to be small, the image constraint at $I(x, y, t)$ with Taylor series can be developed to get:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t. \tag{3.17}$$

From these equations it follows that:

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0 \tag{3.18}$$

which results in

$$I_x V_x + I_y V_y + I_t = 0 \tag{3.19}$$

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:

$$\nabla I^T \cdot \vec{V} = -I_t \tag{3.20}$$

This is an 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:

$$\begin{cases} I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2) \\ \quad \vdots \\ I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n) \end{cases} \tag{3.21}$$

Where $q_1, q_2, \ldots, q_n$ 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 $Av = b$, where

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix} \tag{3.22}$$

This system has more equations than unknowns and thus it is usually overdetermined. The Lucas-Kanade method obtains a compromise solution by the

least squares principle:

$$\begin{aligned} A^T A v &= A^T b \\ \mathrm{v} &= (A^T A)^{-1} A^T b \end{aligned} \tag{3.23}$$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_y(q_i) I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix} \tag{3.24}$$

With this method we can track the interesting points, 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 in 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 calculate using the pinhole model of the camera [27]:

$$\begin{aligned} \omega m &= A[R|t]M \\ \omega \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \end{aligned} \tag{3.25}$$

Where:

- m: is the homogeneous coordinate of the point in the image expressed in pixel.
- M: is the homogeneous coordinate of the correspondent 3D point in the world coordinate frame.
- A: is the camera matrix, or the matrix of intrinsic parameters. It is Composed by f_x, f_y the focal lengths and c_x, c_y the principal point.
- $[R|t]$: is the joint rotation-translation matrix, or matrix of extrinsic parameters. It express the camera motion around the static scene. This matrix denote the coordinate system transformations from 3D world coordinates to 3D camera coordinates. In particular we have to notice that the position C of the camera expressed in world coordinates is $C = -R^{-1}t = -R^T t$.

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

$$z = \frac{l_w f}{l_i} \tag{3.26}$$

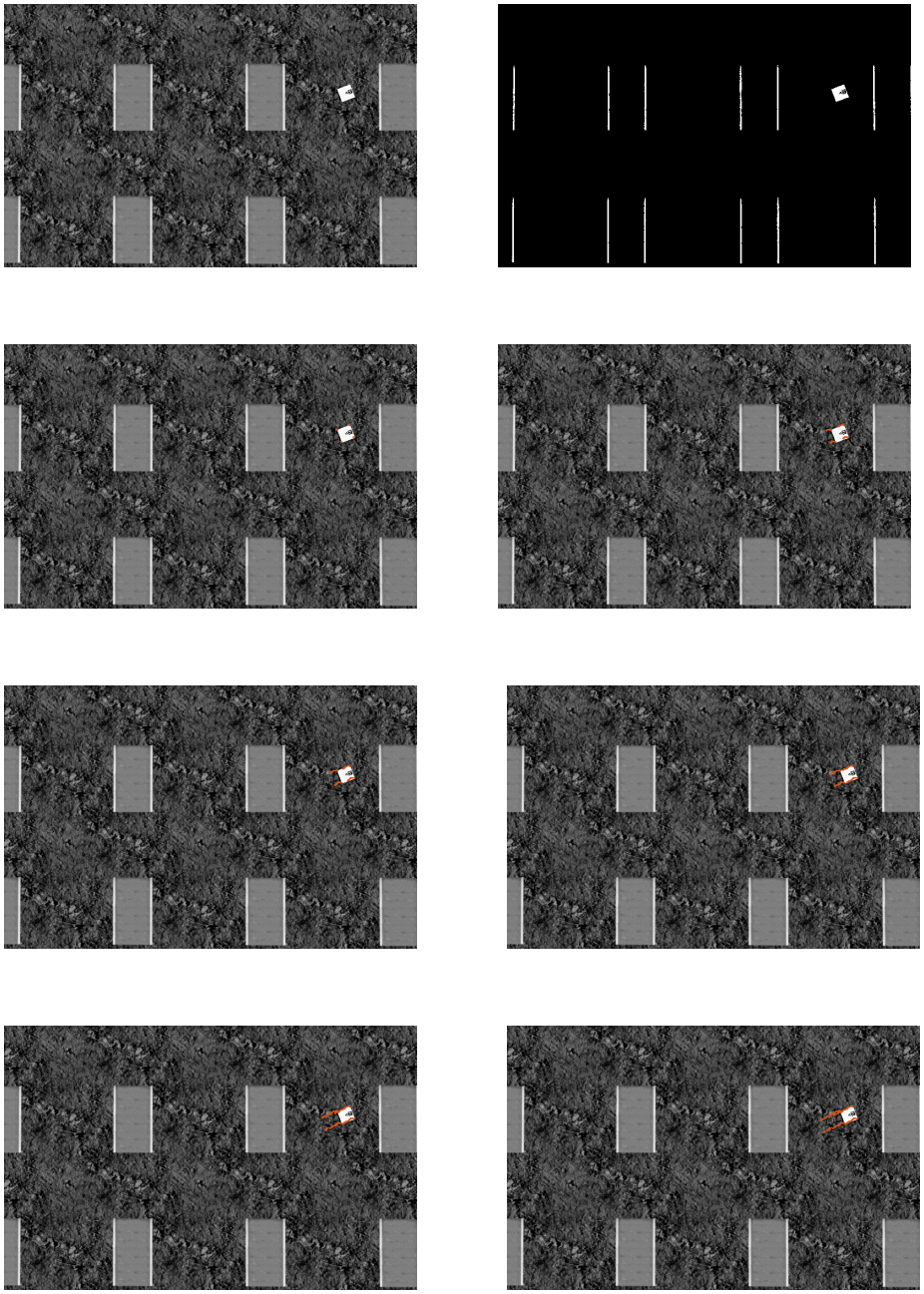


Figure 3.4: A sequence of images where the moving car is detected and tracked. First image is the original image. Then the one after thresholding. Then all the subsequent images where the corners of the platform are tracked.

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 3.5:

- if we have 4 corners there are 6 different distances: 4 of which equal to l_i and 2 $\sqrt{2}l_i$
- if we have 3 corners there are 3 different distances: 2 of which equal to l_i and 1 $\sqrt{2}l_i$

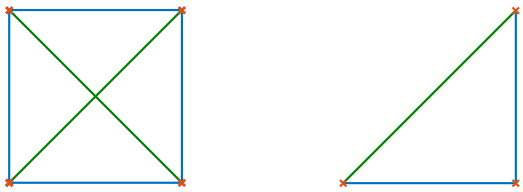


Figure 3.5: Model of the square platform detected on the image. Red crosses corner detected. Blue lines edges with length l_i. Green lines edges with length $\sqrt{2}l_i$

This approximation is not really precise when we see the platform with a camera not perpendicular to the base, but we need just a rough approximation of the height in this first phase, and since the distance between camera and platform are very high, this assumption can hold.

If this depth $z! = 0$ we can solve the system of equation 3.25 finding an unique solution using the following equivalent equations:

$$\begin{aligned} x &= z\frac{u - c_x}{f_x} \\ y &= z\frac{v - c_y}{f_y} \\ \begin{bmatrix} x \\ y \\ z \end{bmatrix} &= R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \end{aligned} \tag{3.27}$$

As we said this method it is not really precise, because we are assuming that the platform surface and the image 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 find 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, so 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 interesting points from frame to frame, calculate direction, velocity and the correspondent 3D pose of the car.

3.2.2 From low altitude

When the quadrotor is closer to the landing platform more features can be seen from the camera. This allow to design a base that helps the measure of the pose of the moving car.
In the final challenge the platform will be as depicted in figure 1.2, while for the first testing another design is considered 3.6, in order to use preexisting algorithm that allow pose estimation with respect to the camera.

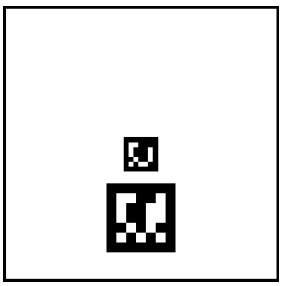
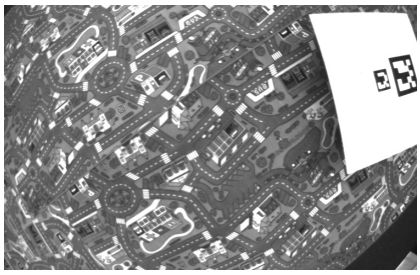


Figure 3.6: Design of the platform, we are using, in which the quadrotor must land on

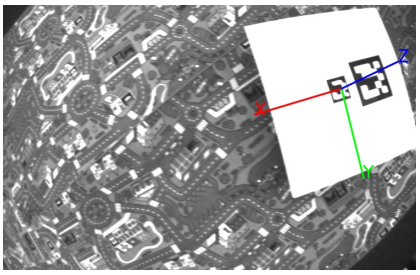
The platform we are using is decorated with Augmented-Reality Tags. AR Tags are planar markers used to easily make virtual objects and animations appear to enter the real world. They also allows video tracking capabilities that calculate the real camera position and orientation relative to the square physical markers in real time.
To reduce the sensitivity to lightning conditions and camera settings planar marker systems typically use bitonal markers (black and white), so there is no need to identify shades of gray, and the decision made per pixel is reduced to a threshold decision. The markers consist of a square black border and a pattern in the interior to an unique identification, when more the one marker is used in the application.

There are several methods to detect and calculate the pose of the markers. Some methods (as ARToolKit [29]), 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 [30]), use an edge based approach, so one need not to deal with thresholds under different illumination conditions and the algorithm can cope with broken sides and missing corners to a certain extent. Both algorithms find on the image the contour of the marker, then the four corners of every potential marker are used to calculate a homography in order

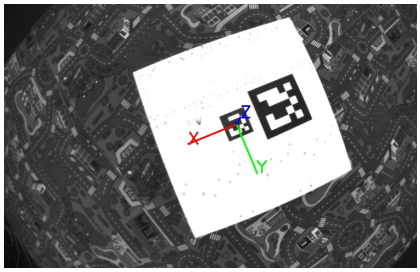
to remove the perspective distortion, solving a Perspective-n-Point problem [28]. Once the internal pattern of a marker is brought to a canonical front view one can sample a grid of $N \times N$ (typically 5×5 or 6×6) points in order to understand the code related to the tag identified, and the orientation of the tag.



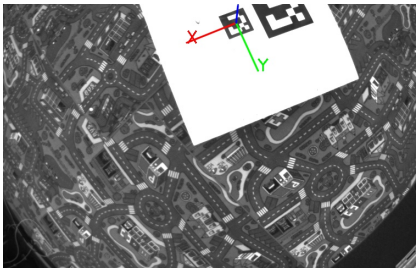
(a) If we are far from the moving platform we have to use the big tag to identify the base.



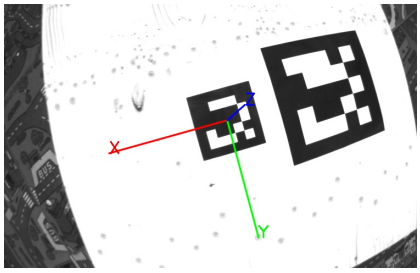
(b) So only when the bigger square is inside the FOV we can detect the center of the base correctly.



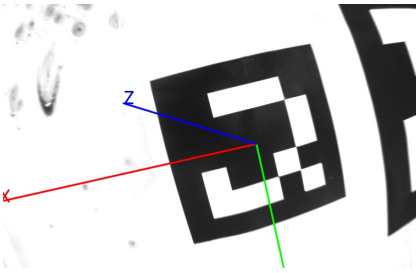
(c) When both the tags are visible we use all the information to have the best position of the master tag.



(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 and further tags leave the FOV.



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

Figure 3.7: 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 with respect to 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.
The equations 3.1 is the general matrix formulation of the system corrupted by a multivariable Gaussian noise $\boldsymbol{w}_k$, but if we consider the covariance matrix Q_k diagonal we can split the equation into:

$$\begin{bmatrix} \dot{x}_k^1 \\ \dot{x}_k^2 \\ \vdots \\ \dot{x}_k^n \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \\ f_2(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \\ \vdots \\ f_n(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \end{bmatrix} + \begin{bmatrix} w_k^1 \\ w_k^2 \\ \vdots \\ w_k^n \end{bmatrix} \tag{3.28}$$

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 generally computed when the variable is predicted with the theoretical model.
For the error update equation 3.2 the concept is the same:

$$\begin{bmatrix} z_k^1 \\ z_k^2 \\ \vdots \\ z_k^m \end{bmatrix} = \begin{bmatrix} h_1(\boldsymbol{x}_{k-1}) \\ h_2(\boldsymbol{x}_{k-1}) \\ \vdots \\ h_m(\boldsymbol{x}_{k-1}) \end{bmatrix} + \begin{bmatrix} v_k^1 \\ v_k^2 \\ \vdots \\ v_k^m \end{bmatrix} \tag{3.29}$$

With v_k^i scalar Gaussian noise with variance r_k^i.
This variance is even more understandable and it is related on the actual error that we are making while measuring the component z_k^i due to measurement limitations.
If these values, used in the update step, are not directly measured, but they derive from other quantities, to calculate their covariance we should propagate the uncertainty:
supposed that in the equation 3.2 $\boldsymbol{z}_k = h(\boldsymbol{x}_k)$ is not a direct measurement of $\boldsymbol{x}_k$ but is a function of other $\boldsymbol{\gamma}_k$, such as:

$$h(\boldsymbol{x}_k) = g(\boldsymbol{\gamma}_k) \tag{3.30}$$

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

In the linear case

$$g(\boldsymbol{\gamma}_k) = \mathrm{A}\boldsymbol{\gamma}_k \tag{3.31}$$

The covariance matrix Σ_g of g is related to $\Sigma_{\boldsymbol{\gamma}}$, the covariance of the variable $\boldsymbol{\gamma}$, by the equation:

$$\begin{aligned} Cov(g) &= Cov(\mathrm{A}\boldsymbol{\gamma}_k) = \mathrm{A}Cov(\boldsymbol{\gamma}_k)\mathrm{A}^\top \\ \Sigma_g &= \mathrm{A}\Sigma_{\boldsymbol{\gamma}}\mathrm{A}^\top \end{aligned} \tag{3.32}$$

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

$$g_i(\boldsymbol{\gamma}_k) \approx g_i(\tilde{\boldsymbol{\gamma}}_k) + \sum_j^n \frac{\partial g_i}{\partial \gamma_j}\Big|_{\tilde{\gamma}_k^j} (\gamma_k^j - \tilde{\gamma}_k^j) \tag{3.33}$$

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:

$$g(\boldsymbol{\gamma}_k) \approx g(\tilde{\boldsymbol{\gamma}}_k) + J\Big|_{\tilde{\gamma}_k} (\gamma_k - \tilde{\gamma}_k) \tag{3.34}$$

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

$$\Sigma_g \approx \mathrm{J}\Sigma_{\boldsymbol{\gamma}}\mathrm{J}^\top \tag{3.35}$$

In our case the update step is defined in 3.15, and it is computed with the methods described in the previous section: we do not have a direct measure of the 3D position and angle θ, but they derive from the measurement of the 2D position of the pixel that corespondent 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

$$g : \mathrm{R}^{6\times 6} \to \mathrm{R}^{2\times 2} \tag{3.36}$$

that is converting the real world coordinate in image coordinate, and its Jacobian matrix

$$J \in \mathrm{R}^{2\times 6} \tag{3.37}$$

To calculate the covariance of the final pose estimate,

$$\Sigma_{RW} \in \mathrm{R}^{6\times 6} \tag{3.38}$$

From the covariance of the image position

$$\Sigma_I \in \mathrm{R}^{2\times 2} \tag{3.39}$$

We need to invert the equation 3.35:

$$\Sigma_{RW} = (\mathrm{J}^\top \Sigma_I \mathrm{J})^{-1} \tag{3.40}$$

In our implementation we do not have a single function g from image pixel (u, v) to 3D coordinate (x, y, z, θ).
The main calculation from coordinates in the image to 6dof pose is done using the openCV function *solvePnP* [31] that returns a pose expressed as 3D position (x, y, z) and Rodrigues orientation [32].
To convert the orientation in Euler angles we convert the Rodrigues angles into a rotation matrix and then the rotation matrix into finally $(roll, pitch, yaw)$ notation.
So in our case we are using a composition of functions and, to propagate the uncertainty through it, we have to calculate the Jacobian of this composition:

$$\begin{aligned}
J_{solvePnP} = J_0 &= \Big[\frac{\partial g}{\partial rodrigues}, \frac{\partial g}{\partial xyzpos}\Big] \\
J_{rodriguesToR} = J_1 &= \frac{\partial rodrigues}{\partial R}, \\
J_{RToEuler} = J_2 &= \frac{\partial R}{\partial euler}, \\
J_{Final} = J &= \Big[\frac{\partial g}{\partial euler}, \frac{\partial g}{\partial xyzpos}\Big] = J_0 \begin{bmatrix} J_1 J_2 & 0 \\ 0 & I \end{bmatrix}
\end{aligned} \tag{3.41}$$

At this point applying 3.40 with the final Jacobian we have the covariance we need in the EKF.

Chapter 4

State machine

This section describes the module that, based on UAV and moving platform odometry estimations, decides in which state the quadrotor is, and which is the desired state that it must reach in order to complete the mission.
This module is implementing a state machine and the flow diagram can be seen in figure 4.1.

It consists in 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 $\rho_{reached}$ and center the final state.

In the following sections we will 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 static 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 is hovering close to the ground, then a vertical takeoff is performed until it reaches a given altitude h (this vertical maneuver is performed anytime in which the pipeline fails and we have to restart the state machine).
Now 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 1.1), we calculate a list of way-points the UAV must reach in order to span the whole surface.
While the quadrotor is moving the camera is collecting information from a large section of the space 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

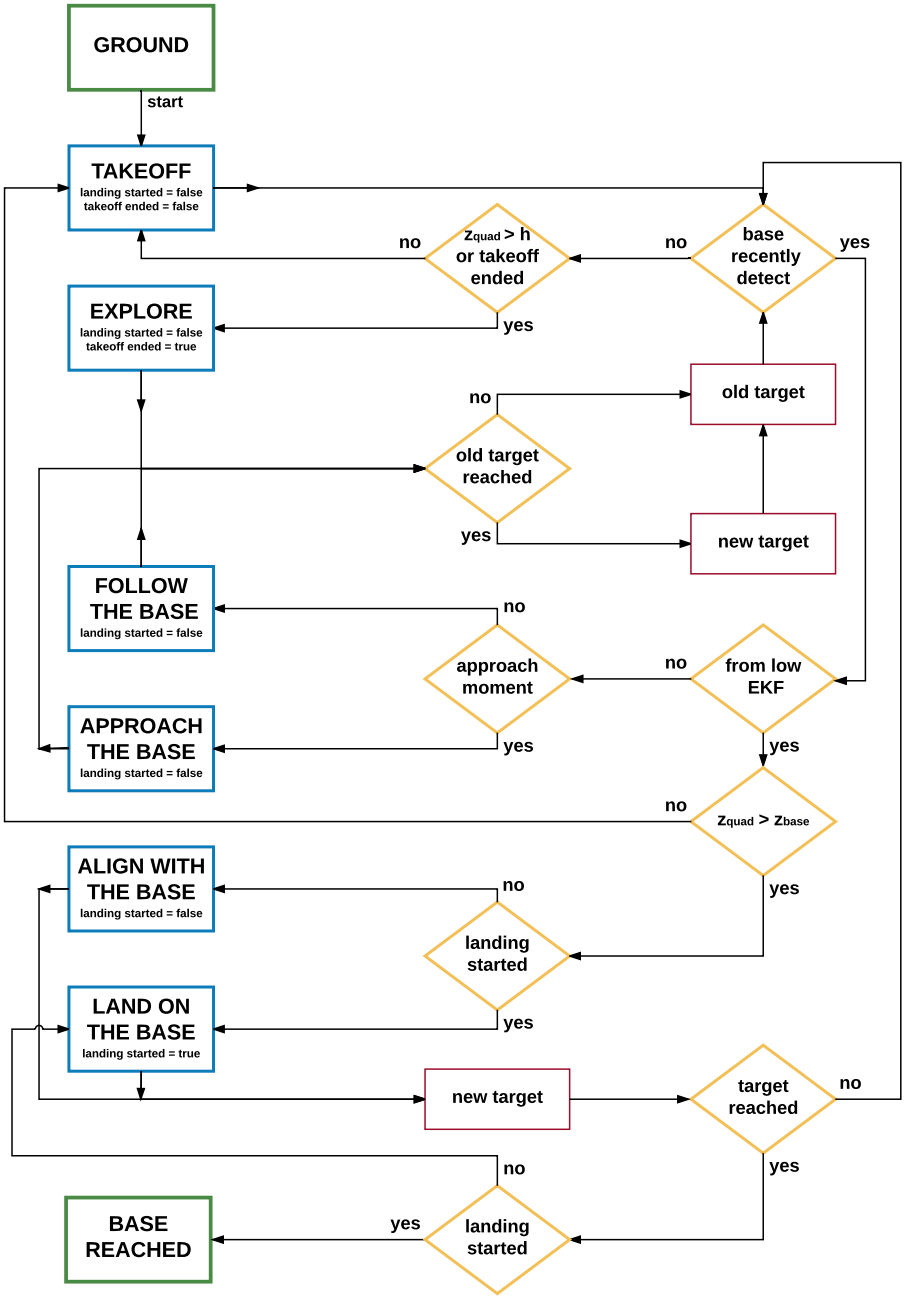


Figure 4.1: State machine flow chart

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 8 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 following the moving platform until the right moment for starting the real landing maneuvers is come.

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

4.2.1 Understand type of movement

From the challenge description 1.1 we know that the car is moving 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 and because we want to proceed with the following phases, and perform the landing maneuver, when the platform is going straight.

To understand the trajectory of the moving base, we collect all the estimated positions of the base coming from the previous module 3, and we perform a linear regression on the last n estimations: the platform is moving in a straight line if the linear regression is a good approximation of the data trends otherwise it driving in 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 he best-fit line that can describe the data as a linear function:

$$y = mx + q$$

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 best of this two fits.
We want find the best best parameters m and q, and to do so we need to have some measure 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:

$$e_i = y_i - (mx_i + q) \tag{4.1}$$

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

$$\sum_{i=1}^{n} e_i^2$$

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

$$e_{tot} = \sqrt{\sum_{i=1}^{n} e_i^2}$$

The parameters m and q of the model, are found where e_{tot}^2 is minimized:

$$\begin{aligned} \frac{e_{tot}^2}{\partial m} &= 0 \\ \frac{e_{tot}^2}{\partial q} &= 0 \end{aligned} \tag{4.2}$$

It is easy to demonstrate that the solution of 4.2 is:

$$m = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \tag{4.3}$$

$$q = \frac{\sum_{i=1}^{n} y_i}{n} - m\frac{\sum_{i=1}^{n} x_i}{n}$$

The platform is moving in the 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.

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:

  $$l_{curve} = \frac{\rho_8 \pi}{4} \tag{4.4}$$

  We know the forward constant velocity of the car v_{tan}, so we can calculate the time in which the platform is performing this curve:

  $$t_{curve} = \frac{l_{curve}}{v_{tan}} \tag{4.5}$$

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

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

$$\tilde{y_i} = \mathcal{N}(y_i, \sigma_e^2) \tag{4.6}$$

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

$$\begin{aligned} <\tilde{e_i}^2> &= <(\tilde{y_i} - (mx_i + q))^2> \\ &= <\tilde{y_i}^2 - 2\tilde{y_i}(mx_i + q) + (mx_i + q)^2> \\ &= <\tilde{y_i}^2> -2<\tilde{y_i}>(mx_i + q) + (mx_i + q)^2 \\ &= \sigma_e^2 + y_i^2 - 2y_i(mx_i + q) + (mx_i + q)^2 \\ &= \sigma_e^2 + e_i^2 \end{aligned} \tag{4.7}$$

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

$$\begin{cases} x_i = x_i \\ y_i = ax_i + b \end{cases}$$

so the theoretical residual,, calculated using 4.1, is:

$$e_i = ax_i + b - (mx_i + q)$$

Of course when we try to calculate the model parameter m and q with 4.3 the result will lead to:

$$\begin{aligned} m &= a \\ q &= b \end{aligned} \tag{4.8}$$

So, obviously, the theoretical residual squares is:

$$e_i^2 = 0$$

And the average residual squares on the measured data is:

$$<\tilde{e_i}^2> = \sigma_e^2$$

The parameter th_{line} is then:

$$th_{line} = \sqrt{\sum_{i=1}^{n} \tilde{e_i}^2} = \sqrt{\sum_{i=1}^{n} \sigma_e^2} = \sigma_e\sqrt{n} \tag{4.9}$$

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

$$\begin{cases} x_i = \rho\cos\theta_i \\ y_i = \rho\sin\theta_i \end{cases}$$

so the theoretical residual, calculated using 4.1, is:

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

To find m and q we use equation 4.3, 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:

$$\begin{aligned} \lim_{n\to\infty} \frac{b-a}{n} \sum_{i=0}^{n} f(x_i) &= \int_a^b f(x)\mathrm{d}x \\ \sum_{i=0}^{n} f(x_i) &\simeq \frac{n}{b-a}\int_a^b f(x)\mathrm{d}x \end{aligned} \tag{4.10}$$

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

$$\begin{aligned} \sum_{i=1}^{n} x_i y_i &= \sum_{i=1}^{n} \rho^2 \cos\theta_i \sin\theta_i \\ &\simeq \frac{n}{\theta_2-\theta_1}\rho^2 \int_{\theta_1}^{\theta_2} \cos x \sin x \mathrm{d}x \\ &= \frac{n}{\theta_2-\theta_1}\frac{\rho^2}{2}\Big[-\cos^2 x\Big]_{\theta_2}^{\theta_2} \\ \sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} \rho\cos\theta_i \\ &\simeq \frac{n}{\theta_2-\theta_1}\rho \int_{\theta_1}^{\theta_2} \cos x \mathrm{d}x \\ &= \frac{n}{\theta_2-\theta_1}\rho\Big[\sin x\Big]_{\theta_2}^{\theta_2} \\ \sum_{i=1}^{n} y_i &= \sum_{i=1}^{n} \rho\sin\theta_i \\ &\simeq \frac{n}{\theta_2-\theta_1}\rho \int_{\theta_1}^{\theta_2} \sin x \mathrm{d}x \\ &= \frac{n}{\theta_2-\theta_1}\rho\Big[-\cos x\Big]_{\theta_2}^{\theta_2} \\ \sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} \rho^2\cos^2\theta_i \\ &\simeq \frac{n}{\theta_2-\theta_1}\rho^2 \int_{\theta_1}^{\theta_2} \cos^2 x \mathrm{d}x \\ &= \frac{n}{\theta_2-\theta_1}\frac{\rho^2}{2}\Big[x+\cos x\sin x\Big]_{\theta_2}^{\theta_2} \end{aligned} \tag{4.11}$$

In our case we consider pieces of curve with length l_{curve} (defined in 4.4), that corresponds to a circumference arch with:

$$\rho = \rho_8 \qquad \theta_i \in \left[0, \frac{\pi}{4}\right] \tag{4.12}$$

We can now calculate the approximate values of m and q using 4.3 4.11 4.12:

$$\begin{aligned} m &= \frac{n\rho_8^2\frac{n}{\pi} - \rho_8\frac{n2\sqrt{2}}{\pi}\rho_8\frac{n2(2-\sqrt{2})}{\pi}}{n\frac{n\rho_8^2(2+\pi)}{2\pi} - (\rho_8\frac{n2\sqrt{2}}{\pi})^2} \\ &= \frac{2\pi - 16\sqrt{2} + 16}{\pi^2 + 2\pi - 16} \end{aligned} \tag{4.13}$$

$$\begin{aligned} q &= \frac{\rho_8\frac{n2(2-\sqrt{2})}{\pi}}{n} - m\frac{\rho_8\frac{n2\sqrt{2}}{\pi}}{n} \\ &= \rho_8\frac{4 - 2\sqrt{2}(m+1)}{\pi} = \rho_8\bar{q} \end{aligned}$$

Now we should calculate the theoretical residual square e_i^2, but in this case we can compute the algebraic average residual square $< e_i^2 >$, using again the approximations 4.11, the values calculate in 4.13, and averaging over the n samples we consider:

$$\begin{aligned} < e_i^2 > &= \frac{1}{n}\sum_{i=1}^{n}\Big(\rho_8\sin\theta_i - (m\rho_8\cos\theta_i + q)\Big)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\rho_8^2\xi = \rho_8^2\xi \end{aligned} \tag{4.14}$$

In particular with θ having values like in 4.12 we lead with:

$$\xi = \frac{\pi - 2 + m^2(\pi+2) + 2\pi\bar{q}^2 - 4m - 8\bar{q}(2 - \sqrt{2} + m\sqrt{2})}{2\pi} \tag{4.15}$$

Finally we calculate

$$< \tilde{e_i}^2 > = < e_i^2 > + \sigma_e^2 = \rho^2\xi + \sigma_e^2$$

The parameter th_{curve} is then:

$$th_{curve} = \sqrt{\sum_{i=1}^{n}\tilde{e_i}^2} = \sqrt{\sum_{i=1}^{n}\rho^2\xi + \sigma_e^2} = \sqrt{n}(\sigma_e + \rho\sqrt{\xi}) \tag{4.16}$$

The 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

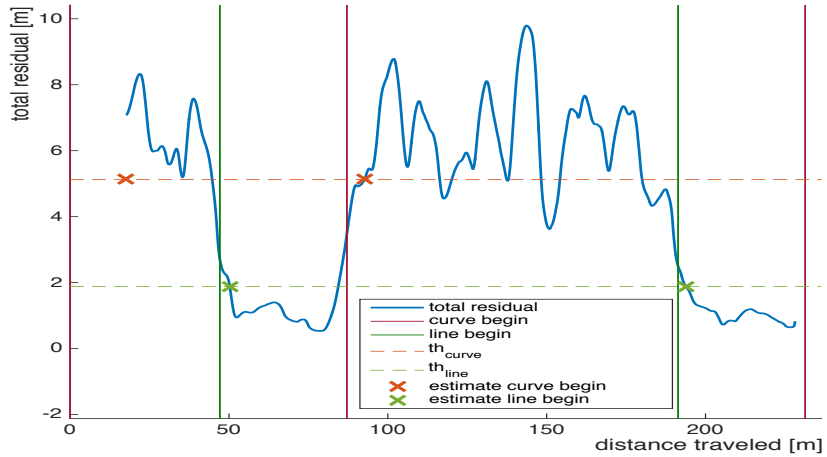


Figure 4.2: Evolution of the total residual during this first phase (in blue). The vertical lines are the real moments in which the car changes movement types: green a linear phase starts, red a circular phase begins. The horizontal lines are the thresholds for the detection of the two different regimes.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

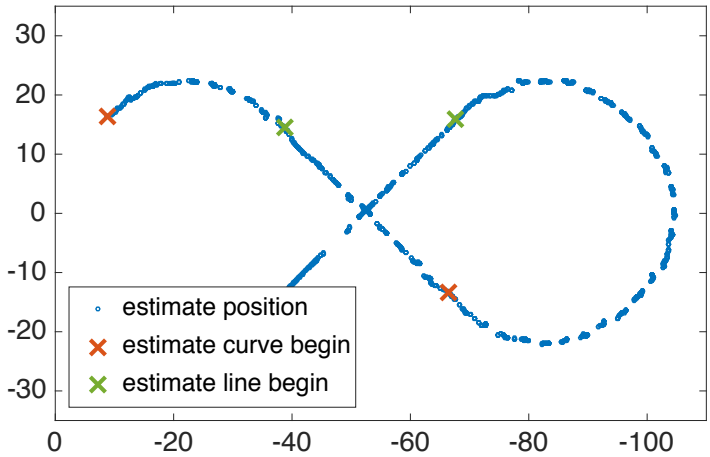


Figure 4.3: Map of the estimated positions of the platform in blue. The crosses are the moments in which the algorithm understands the change. Red crosses from line to curve. Green crosses from curve to line.

the graph. Furthermore the point of regime change can be seen both in figure 4.2 and in the map 4.3 in which also all the estimate positions of the base are plotted.

We can see from these graphs that with the method explained 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 finding the switching points with high 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.
If th_{line} and th_{curve} will result too close it is always possible to consider a curve with longer length l_{curve}: this will increase much more the letter threshold w.r.t the former, but also will increase the time t_{curve} to understand the type of movement.

4.2.2 Calculate future position

Now knowing the platform regime of movement at a specific time we can estimate correctly 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 at the moment t_0 the car is at position (x_0, y_0) with a direction angle of θ_0 and forward velocity of v_{tan}, so at time $t_1 = t_0 + t_s$ seconds the car will be at position (x_1, y_1) with an angle θ_1, and it has traveled $v_{tan}t_s$.

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

$$\begin{cases} x_1 &= x_0 + v_{tan}t_s\cos\theta_0 \\ y_1 &= y_0 + v_{tan}t_s\sin\theta_0 \\ \theta_1 &= \theta_0 \end{cases} \tag{4.17}$$

- When a movement in the circumference is detected we have to perform some calculations in order to find the final state of the platform.
  First of all we use the relation

$$l_{curve} = v_{tan}t_s = \rho_8|\beta_s|$$

  Where β_s is the angle spanned by the platform in t_s seconds:

$$|\beta_s| = \frac{v_{tan}t_s}{\rho_8} \tag{4.18}$$

  The final angle will be:

$$\theta_1 = \theta_0 + \beta_s \tag{4.19}$$

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

– direction θ_{chord} found as bisection between θ_0 and θ_1

$$\theta_{chord} = \theta_0 + \frac{\theta_0 + \theta_1}{2} \tag{4.20}$$

– length l_{chord}, found with the chord theorem:

$$l_{chord} = 2\rho_8 \sin \frac{|\beta_s|}{2} \tag{4.21}$$

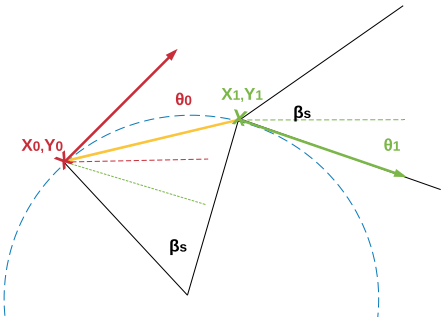


Figure 4.4: In red position of initial state at time t_0. In green final estimate state at time t_1. In yellow the chord between the two states with length l_{chord} and orientation θ_{chord}.

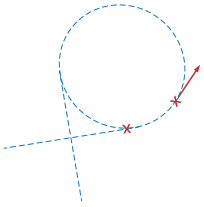
Now we have all the elements to calculate the final point (x_1, y_1), but in order to properly find it we have to resolve an other last problem: both β_s and $-\beta_s$ span a curve of length $v_{tan}t_s$, and due to the symmetry of our trajectory is impossible to know before which angle is the right one.
What we can do is calculate both the two possible final states using 4.18 4.20 4.21:

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

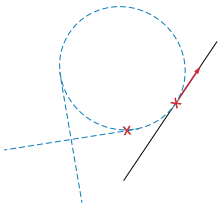
$$\begin{cases} x_1^b &= x_0 + l_{chord} \cos\left(\theta_0 - \frac{|\beta_s|}{2}\right) \\ y_1^b &= y_0 + l_{chord} \sin\left(\theta_0 - \frac{|\beta_s|}{2}\right) \\ \theta_1^b &= \theta_0 - |\beta_s| \end{cases}$$

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 $t_{-\alpha} < t_0$. For sure the state with smaller distance will be the right final state, because the wrong one leads to a position further away.

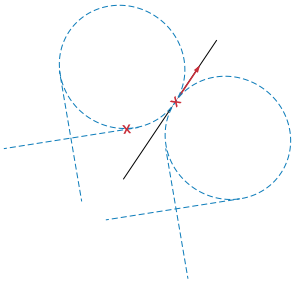
The images 4.5 summarize the passages we perform to find the right final state just explained.



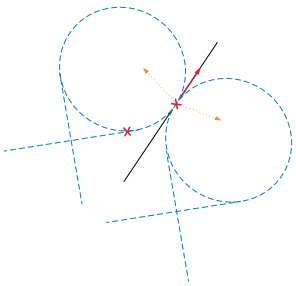
(a) Red cross with arrow: state a t_0. Red arrow: current velocity vector. Red cross: previous position a $t_{-\alpha}$.



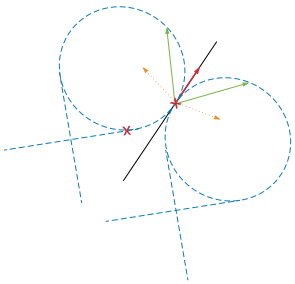
(b) Black line: direction of the velocity vector, with angle θ_0. We have symmetry with respect to this axes.



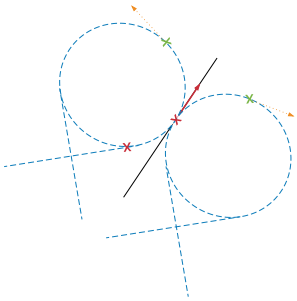
(c) Blue lines: real and symmetric path. We do not know which of the two trajectories is correct.



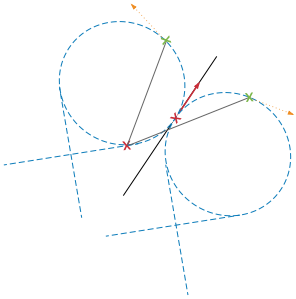
(d) Yellow arrows: estimated future angles θ_1^a and θ_1^b that the base will assume at time t_1.



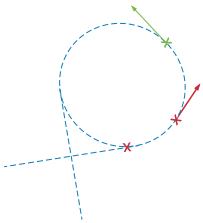
(e) Green arrows: segment from state at t_0 and possible states at t_1: with angles θ^a_{chord} and θ^b_{chord} and length l_{chord}



(f) Green crosses: future possible states of the platform at positions (x_1^a, y_1^a) and (x_1^b, y_1^b) .



(g) Dark grey lines: distances from previous position at $t_{-\alpha}$ and possible future positions at t_1.



(h) Green cross with arrow: future state selected taking the position with minimum distance.

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

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 following the moving car. It is noticeable the subdivision of point calculates with the linear model (green stars) and with the circular one (red stars).

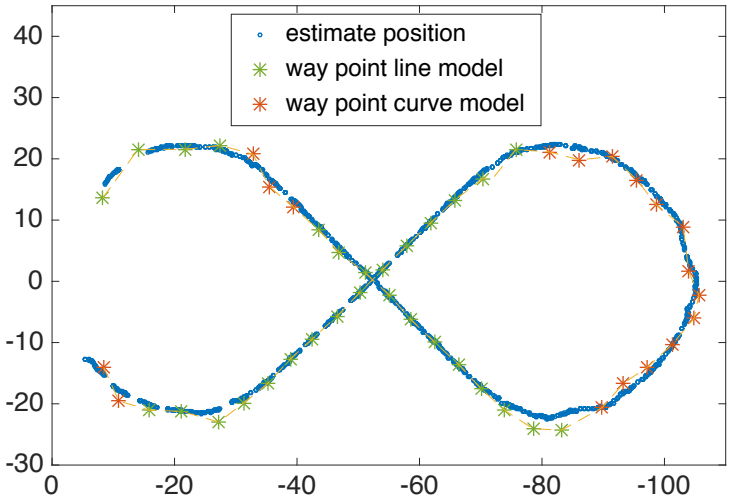


Figure 4.6: Map of the estimated positions of the platform in blue. Stars positions in which the quadrotor should go to following the base until a proper moment to proceed with the following stage is detected. The green points are calculated with the linear model while the red ones with the circular.

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 the landing even a 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:

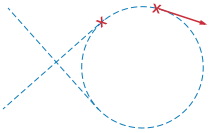
  - we know the orientation θ_{line} of the straight line just finished: the platform just changed from line regime to curve and we have saved the inclination m of the best linear approximation found while it was moving in line.
  - given the point of change between line and curve, the future point will be in the circumference after an angle of $\left|\frac{3\pi}{2}\right|$.
  - from equations 4.20 4.21 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:

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

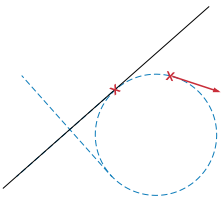
    $$\begin{cases} x^b_{intersection} &= x_{changing} + \sqrt{2}\rho_8 \cos\left(\theta_{line} - \frac{3\pi}{4}\right) \\ y^b_{intersection} &= y_{changing} + \sqrt{2}\rho_8 \sin\left(\theta_{line} - \frac{3\pi}{4}\right) \\ \theta^b_{intersection} &= \theta_{line} - \frac{3\pi}{2} \end{cases}$$

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

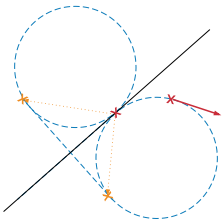
  The images 4.7 summarize all the passages we perform to find the right intersection point just explained.



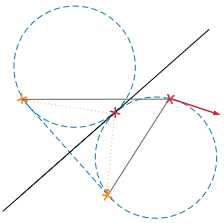
(a) Red cross with arrow: state a t_0. Red arrow: current velocity vector. Red cross: changing point from line to curve.



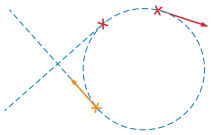
(b) Black line: direction of the line sector just finished. The direction is taken as the slope of the best linear fit found in the previous regime.



(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 becouse 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.

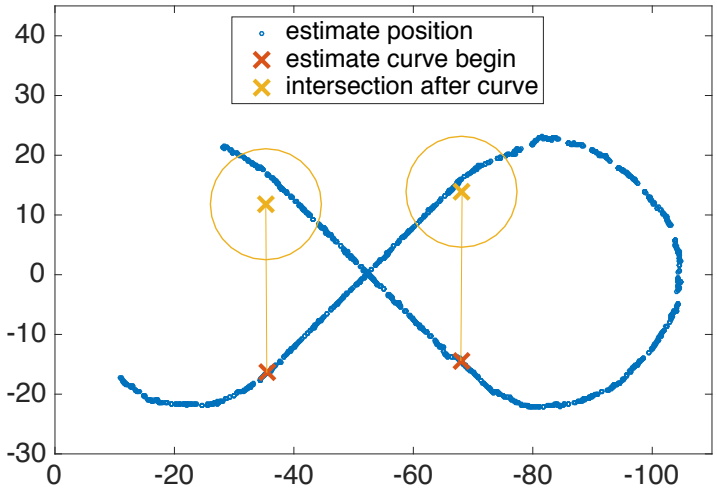


Figure 4.8: Map of the estimated positions of the platform in blue. Yellow crosses the position where the quadrotor should go in order to intersect the platform when is about to start a line phase.

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 4.17. 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:

$$\begin{cases} vx_1 &= v_{tan} \cos\theta_0 \\ vy_1 &= v_{tan} \sin\theta_0 \end{cases} \tag{4.22}$$

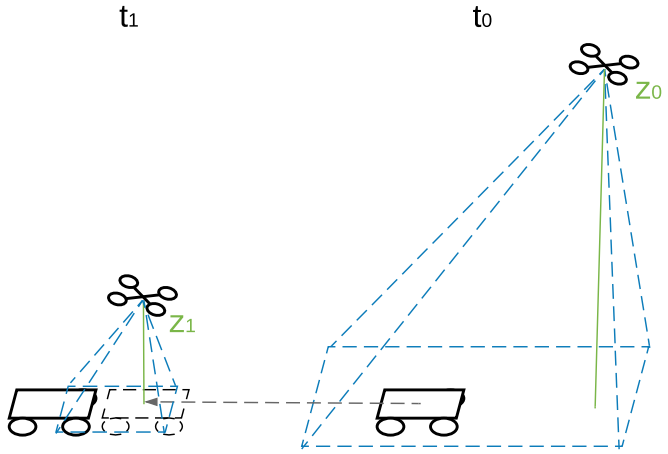
While the final altitude:

$$z_1 = \alpha z_0 + (1 - \alpha) z_{0,target} \tag{4.23}$$

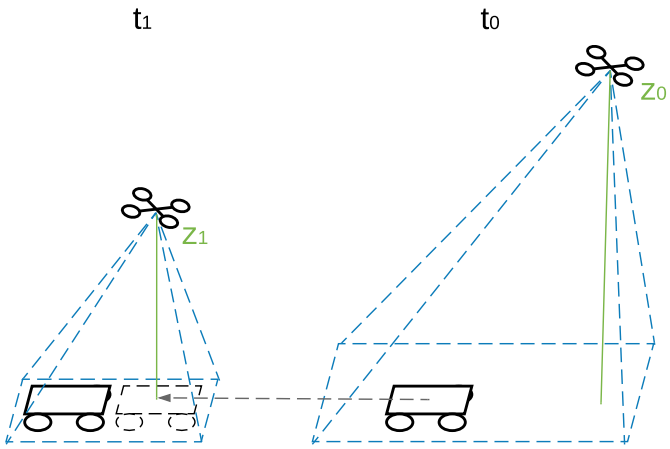
Where $\alpha < 1$ is a parameter to be tuned in order to have a right balance between fastness of this stage and aggressiveness of the maneuver.
As a matter of fact if the quad approaches too quickly the target, it is very easy to lose the platform from the fov of the camera: the closer we are to the 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 he 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 relay 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 now 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 an maximum relative speed:

$$v_{max,rel} = v_{max,quad} - v_{base} \tag{4.24}$$

In a time:

$$t = \frac{d}{||v_{max,rel}||} \tag{4.25}$$

We can see from equation 4.24 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 3.8. 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}.

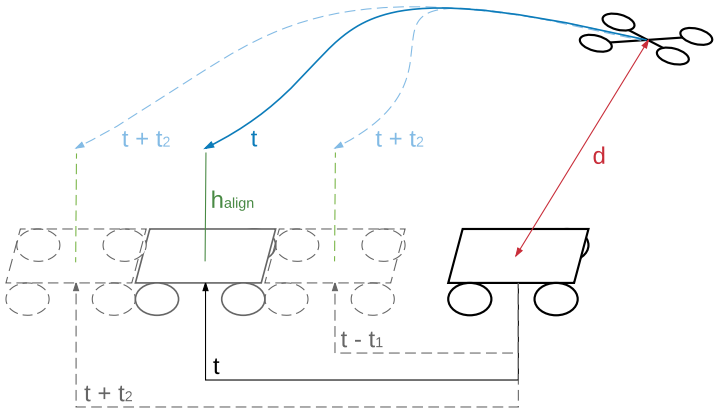


Figure 4.10: The scheme synthesizes the algorithm performed in this stage to find the possible final states in which the quadrotor should reach in order to intercept the base.

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:

$$z(t) = z_{quad,0} + vz_{quad,0}t + \frac{(c_{quad,z} - g)t}{2} \tag{4.26}$$

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 $vz_{quad,0}$ really little, so it can be neglect. We can then easily calculate the time t_{blind}:

$$t_{blind} = \sqrt{\frac{2h_{blind}}{g - c_{quad,z}}} \tag{4.27}$$

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 :

$$\begin{cases} x_{base}(t_{blind}) & = x_{base}(0) + v_{tan}t_{blind}\cos(\theta_{base}) \\ y_{base}(t_{blind}) & = y_{base}(0) + v_{tan}t_{blind}\sin(\theta_{base}) \end{cases} \tag{4.28}$$

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 $h_{blind} \to 0$ also $t_{blind} \to 0$ and so the quad arrives at the intersection point with the moving base).

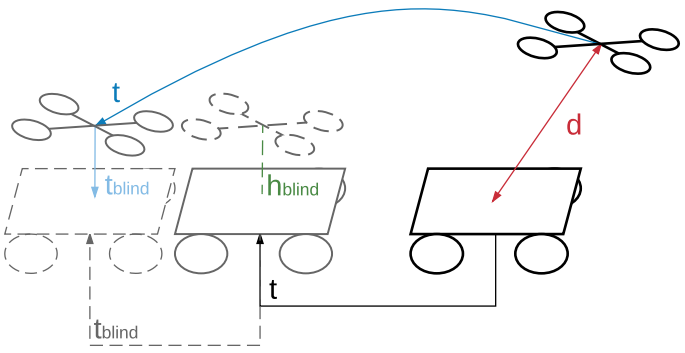


Figure 4.11: The scheme synthesizes the concept of the final blind landing.

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:

$$imu_{filt}(t_k) = (1 - e^{-\frac{t_k - t_{k-1}}{\tau_{imu}}}) imu_{raw}(t_k) + e^{-\frac{t_k - t_{k-1}}{\tau_{imu}}} imu_{filt}(t_{k-1}) \tag{4.29}$$

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 τ_{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.

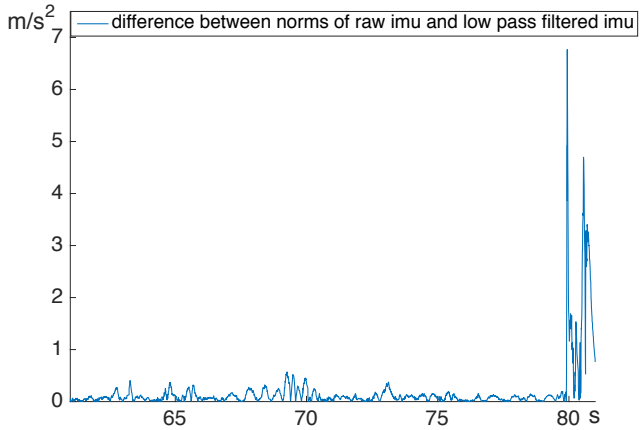


Figure 4.12: Data used by the bump detector: difference between the norms of the raw data from the imu and the filtered version of the same. The difference is growing really fast only when the UAV bumps on the surface.

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 [33]: 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.

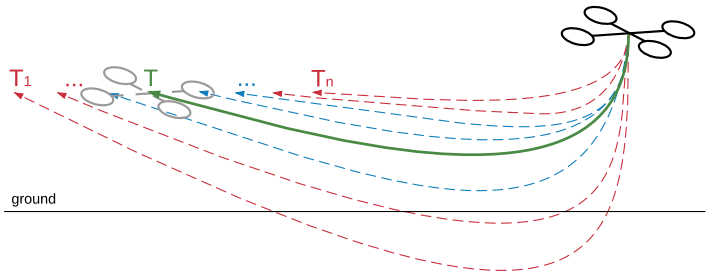


Figure 5.1: The scheme synthesizes the concept of multiple possible trajectory generated and then pick the best one. The red trajectory are unfeasible (state or input unfeasible), the green trajectory is the best solution found.

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 [17] 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.

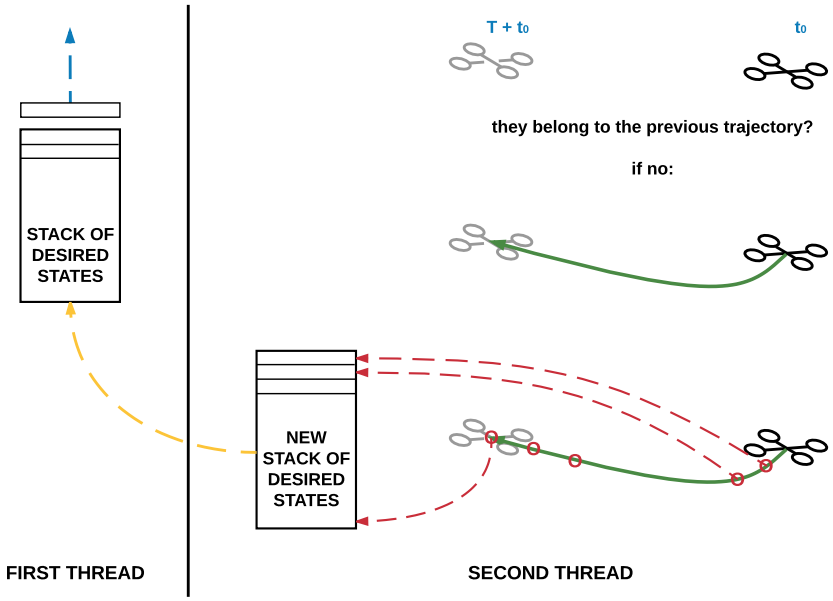


Figure 5.2: Scheme of tasks subdivision between the two threads.

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:

$$\begin{cases} \ddot{\boldsymbol{r}} = \boldsymbol{g} + \boldsymbol{R}_{WB}\boldsymbol{c} \\ \dot{\boldsymbol{R}}_{WB} = \boldsymbol{R}_{WB}\boldsymbol{w}_{\hat{WB}} \end{cases} \tag{5.1}$$

Where

$$\boldsymbol{w}_{\hat{WB}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \qquad \boldsymbol{c} = \begin{bmatrix} 0 \\ 0 \\ c \end{bmatrix} = \boldsymbol{e}_3 c \tag{5.2}$$

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:

$$c^2 = ||\boldsymbol{c}||^2 = ||\ddot{\boldsymbol{r}} - \boldsymbol{g}||^2 \tag{5.3}$$

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

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

$$\boldsymbol{j} = \dddot{\boldsymbol{r}} = \dot{\boldsymbol{R}}_{WB}\boldsymbol{c} + \boldsymbol{R}_{WB}\dot{\boldsymbol{c}} \tag{5.5}$$

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

$$\begin{bmatrix} \omega_1 \\ \\ \omega_2 \end{bmatrix} = \frac{1}{c} \begin{bmatrix} 1 & 0 & 0 \\ \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{R}_{WB}^T \boldsymbol{j} \tag{5.6}$$

Using these equations the inputs of the system are defined with a degree of freedom in ω_3.

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

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

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

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 $\boldsymbol{\omega}$ are then recovered from $\boldsymbol{j}$ 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:

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

subject to the dynamics:

$$\begin{aligned} j_k &= \dddot{r}_k \\ z_k &= \begin{bmatrix} r_k \\ \dot{r}_k \\ \ddot{r}_k \end{bmatrix} = \begin{bmatrix} 1 & dt & \frac{dt^2}{2} \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} z_{k-1} + \begin{bmatrix} \frac{dt^3}{6} \\ \frac{dt^2}{2} \\ dt \end{bmatrix} j_{k-1} \\ z_0 &= z(0) \\ z_N &= z(T) \end{aligned} \tag{5.9}$$

and respecting the constraints:

$$A \begin{bmatrix} r_k \\ j_k \end{bmatrix} \leq b \tag{5.10}$$

The solution of this optimal control problem can be found in close form with Pontryagin's minimum principle. In the paper [17] 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:

$$J = \sum_{k=0}^{N-1} ||\boldsymbol{j}_k||^2 \tag{5.11}$$

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):

$$c_k^2 ||\boldsymbol{\omega}_k||^2 = \left|\left| c_k \begin{bmatrix} \omega_{1,k} \\ \omega_{2,k} \end{bmatrix} \right|\right|^2 = \left|\left| c_k \frac{1}{c_k} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{R}_{WB,k}^T \boldsymbol{j}_k \right|\right|^2 \leq ||\boldsymbol{j}_k||^2 \tag{5.12}$$

5.1.4 Constraints

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

$$\begin{aligned} 0 < c_{min} \leq c \leq c_{max} \\ ||\boldsymbol{\omega}|| \leq \omega_{max} \end{aligned} \tag{5.13}$$

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

$$\begin{aligned} c_{min}^2 \leq c^2 \leq c_{max}^2 \\ c_{min}^2 \leq ||\ddot{\boldsymbol{r}} - \boldsymbol{g}||^2 \leq c_{max}^2 \end{aligned} \tag{5.14}$$

For the body rates

$$||\boldsymbol{\omega}|| = \begin{bmatrix} \omega_1 & \omega_2 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} + \omega_3^2 = \begin{bmatrix} \omega_1 & \omega_2 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} \leq \frac{1}{c}||\boldsymbol{j}|| \leq \omega_{max} \tag{5.15}$$

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:

$$\max_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{1}_z g)^2 > c_{max}^2 \quad \forall i \in \{x, y, z\} \tag{5.16}$$

$$\min_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{1}_z g)^2 < c_{min}^2 \quad \forall i \in \{x, y, z\} \tag{5.17}$$

where $\mathbb{1}_z$ is equal to 1 if we are considering the z axis otherwise it is 0.
On the other hand trajectory is surely feasible if:

$$\sum_i \max_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{1}_z g)^2 \leq c_{max}^2 \quad i \in \{x, y, z\} \tag{5.18}$$

$$\sum_i \min_{k=[0,N]} (\ddot{r}_{i,k} - \mathbb{1}_z g)^2 \geq c_{min}^2 \quad i \in \{x, y, z\} \tag{5.19}$$

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:

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

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. This 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:

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

- 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 filter with low pass filter:

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

- 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 $\boldsymbol{c}$ applied and the rotation of the quadrotor $\boldsymbol{R}_{WB}$:

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

  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 λ_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 [34]. 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 :

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

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

Using these dynamics we need a derivative more in order to express the input in therms 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 it is not correct for the initial state), while the other values of the initial condition are calculated as int the previous section's algorithm.

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

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

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 $\boldsymbol{r}$), and the yaw angular acceleration for minimizing the input relative (in the paper [34] 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:

$$\begin{aligned} &min \;\; \boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} + \boldsymbol{h}^T \boldsymbol{x} \\ &s.t. \;\; \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \end{aligned} \tag{5.28}$$

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 [17] 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 α 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

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

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 ω_{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 thee 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 frequent replanning

The main drawback of the algorithm used is that, in theory, it is possible to replan the trajectory at each control loop: in a MPC style, at each cycle, we calculate a trajectory from the initial state state to the final one and communicate just the first desired state to the high level control, repeating this procedure at the next loop.
In practice, using this algorithm, the continuous replanning is not possible: when we pass the first desired state at the high controller the difference in position, velocity and acceleration are too tiny to actually generate a response from the quadrotor and start a movement. The outcome of this process is that the UAV does not move, as it should, and at the following control cycle the initial condition are only slightly changed, so the quadrotor results static in the initial position.

This behaviour usually does not happen when we have to perform a trajectory in which the altitude is decreasing (small thrust required), but it is a great issue

when the z position should remain equal or increasing.
This is why we do not perform a replan at each loop but only when it is necessary so when the final desired state has changed or the quadrotor is not tracking the trajectory.

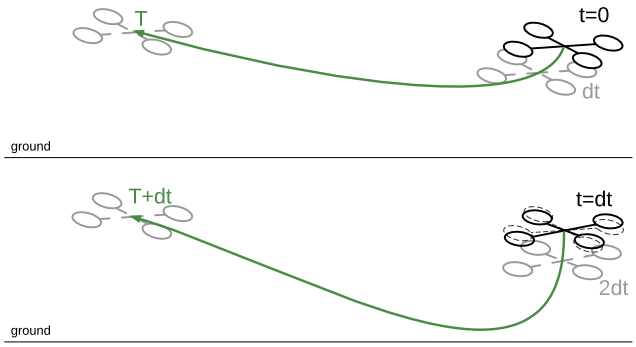


Figure 5.3: The scheme synthesizes the concept too frequent replanning. The quadrotor at time dt should be in a position but the controller are not able to bring the UAV there. At the next loop the state of the quad is almost not changed, so it remains fixed in its initial state.

5.3.3 Too short final time

The rapid trajectory algorithm has another issue: when the final time T is too short all the trajectory calculate result indeterminable. This problem is due to the method used to check the feasibility with respect to the input, 5.1.5, but to overcame this problem we can reduce the threshold for which the algorithm stop to recursively control if a piece of the trajectory is feasible.
In this way the generation of the trajectory is slower, but we are able to find feasible trajectory for shorter time.

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, the simulation environment and we report the main results achieved during these experiments.

6.1 Real world hardware

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 6.2.
These platforms are lightweight ($\tilde{5}00$ grams) and safe for operation in close 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 m/s during vision-based flight.
The quadrotor used is equipped with:

- an inertial measurement unit (IMU) that reports 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 down looking regular camera, used to detect and tracking the moving platform

– a forward looking camera with fish eye lens, used for the quad self state estimation. The fish eye is used in order to have a bigger field of view and be able to track enough features in every configuration.

TODO PHOTO

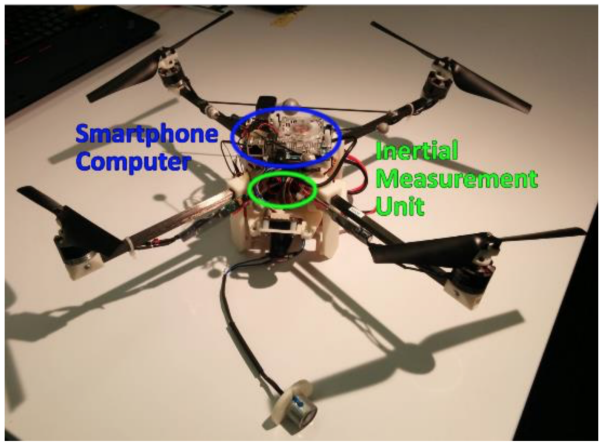


Figure 6.1: The UAV used during the experiments

6.1.2 Moving platform

During the experiments we are using Jackal UGV as moving platform.
Jackal is a small field robotics research platform produced by Clearpath Robotics [35]. It has an onboard computer, GPS, IMU and it is fully integrated with ROS. It can reach a maximum speed of $2m/s$ that is perfect for testing with condition similar to the final challenge.
Over the UGV we have installed a wooden base $1m \times 1m$ were we can attached both the textures 1.2 or 3.6.

Figure 6.2: The 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 [36]: 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 are using RotorS simulator [37]: a UAV gazebo simulator that provides some multirotor models among which there is the AscTec Hummingbird, very similar to the quadrotor we are using in the real world experiments. All quadrotor can be provided with many sensors such as IMU, cameras etc.
To simulate the moving platform we are using the ROS package [38] that allow 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 w.r.t. the one explained in chapter 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 σ^2 variance.

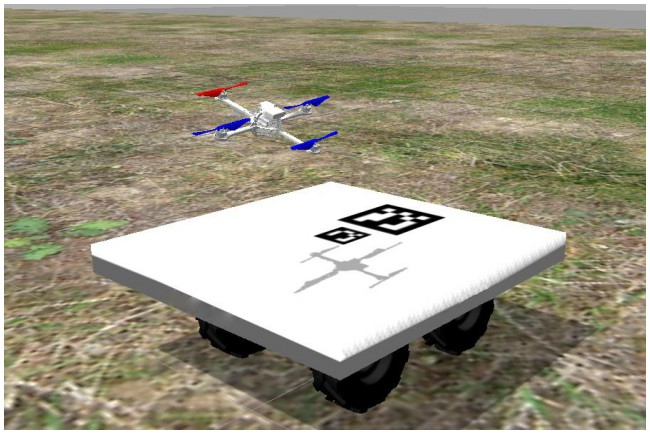


Figure 6.3: The UAV and UGV used during the experiments in simulation

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.
In the final parts of the framework,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 have taken data sets in which we are running 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 motion capture system [39]. In this way we can compare the 2 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 were holding the quadrotor by hand and we were moving inside the flyingroom simulating a square trajectory 6.9.
In particular the first images show the position 6.4, the orientation 6.5 and the velocity 6.6 estimations from the two versions of SVO compared with the optitrack.

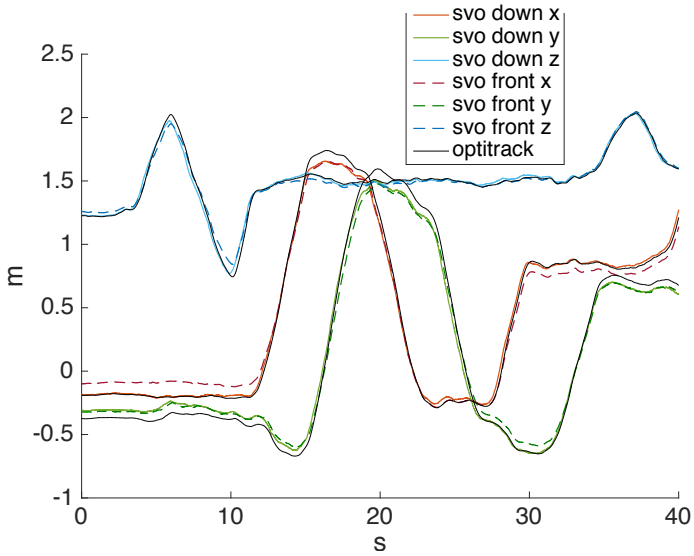


Figure 6.4: Comparison between SVO position estimations with the front looking camera (dashed lines), down looking camera (solid) and optitrack (black)

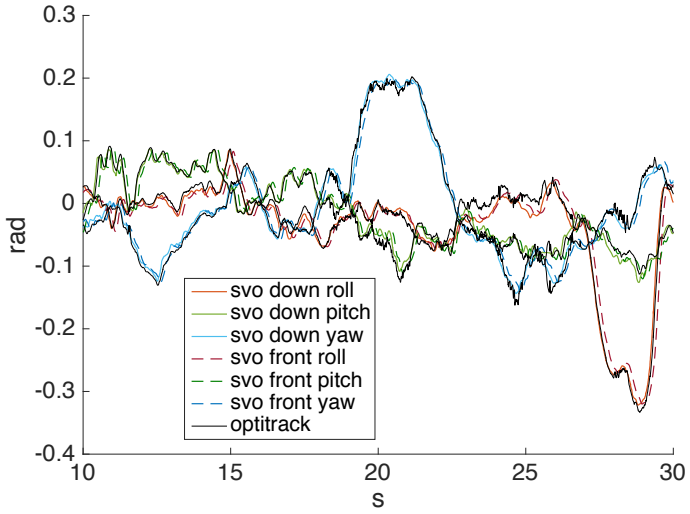


Figure 6.5: Comparison between SVO orientation estimations with the front looking camera (dashed lines), down looking camera (solid lines) , and optitrack (black lines). The orientation data from the optitrack are low pass filtered in order to eliminate the high frequency noise.

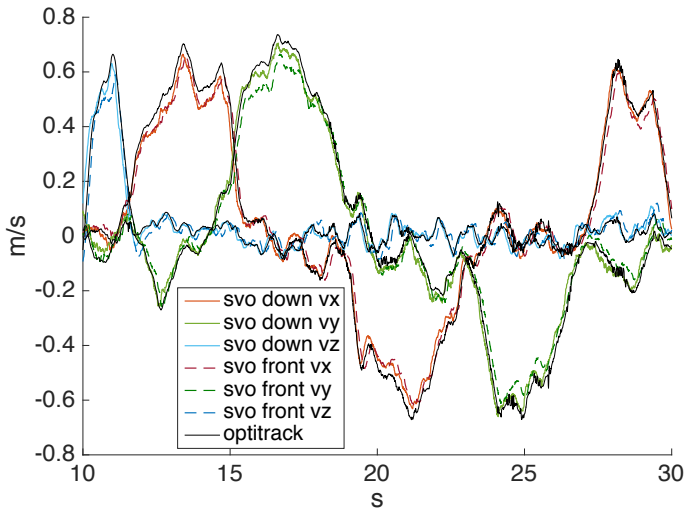


Figure 6.6: Comparison between SVO position estimations with the front looking camera (dashed lines), down looking camera (solid lines), and optitrack (black lines). The velocity data from the optitrack are low pass filtered in order to eliminate the high frequency noise.

We can see that both the versions of our estimation framework are calculating a reliable estimation of the quadrotor pose and velocity. In order to evaluate the precision of the two estimations, see figure 6.7. It shows the average position error between the two versions of SVO: error generally below $10cm$ with RMSE

of 4.5cm for the down looking camera and 6cm for the front looking one.

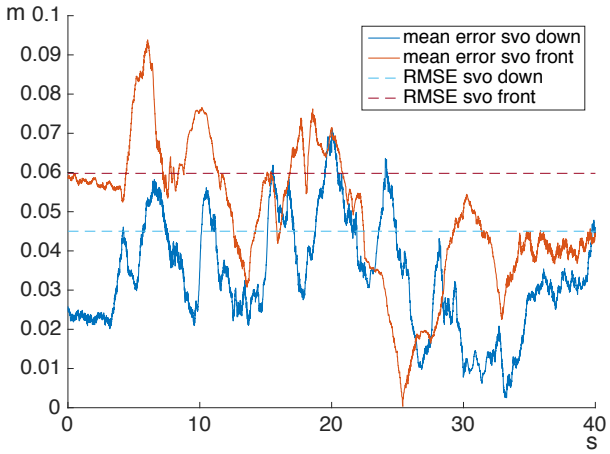


Figure 6.7: Mean error between the 3D position estimations of the two versions of SVO and the ground truth. Blue line is the error with the down looking camera, the red line with the front looking one. The correspondent dashed lines are the RMSE for the two estimations.

This error is larger then expected. In particular the graphs 6.8 shows the percentage error of the position estimations, related to the maximum distance traveled in each direction. We can see that the error is below 6% in x, y and 8% in z that correspond to a general absolute error smaller than 10cm.

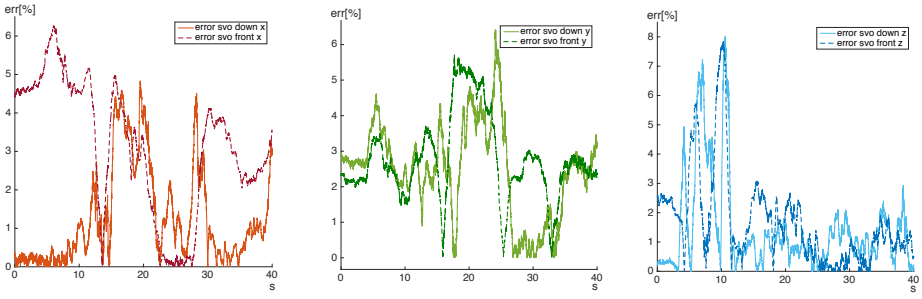


Figure 6.8: Percentage error, in each dimension, between the two version of SVO and the ground truth given by the optitrack. The quad traveled more or less 2m in x,y direction and 1m in z.

From the image of the 3D trajectory 6.9 we can see that the main problem is the scale factor, not drifting in the state estimation. The scale factor is estimated at the beginning, tracking features of the images and setting their depth using external means, other then the images (default depth or laser). If there is an error in this initialization then all the world is scaled and will result bigger or smaller then the reality.

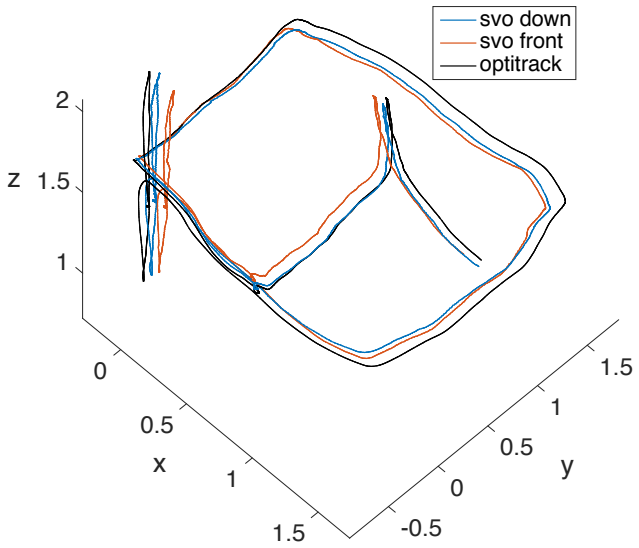


Figure 6.9: Comparison between SVO position estimation in 3D world. The blue line is the estimation with down looking camera, the red line with front looking one and the black line the ground truth given by the optitrack

6.3.2 Drifting

We made other experiments to understand if the current version of frontlooking-fisheye-SVO is good enough to fly with it.
We performed several manual flights in the flyingroom, with the same results: generally SVO is estimating a correct and precise state of the quad, but sometimes it occurs that the state estimation is drifting w.r.t the real world.
For example the data set showed in figure 6.10 is related to a flight in which three times the SVO estimation drifted with respect to the real world in all three axes. We Highlighted in the figure these moments.

The reasons for this behavior can be related to the poverty of features that the vertical walls in the flyingroom has. As a matter of fact if there are no enough distinctive features the visual odometry can drift. TODO ???

6.3.3 Very fast flight

In the final challenge the moving car will move at $15\frac{km}{h}$ that means $4.17\frac{m}{s}$. IN order the quadrotor to be able to follow and land on the platform it must flight faster then this velocity.
We made some experiments to understand if the state estimation with the front looking camera it is precise at this high speed. TODO complete

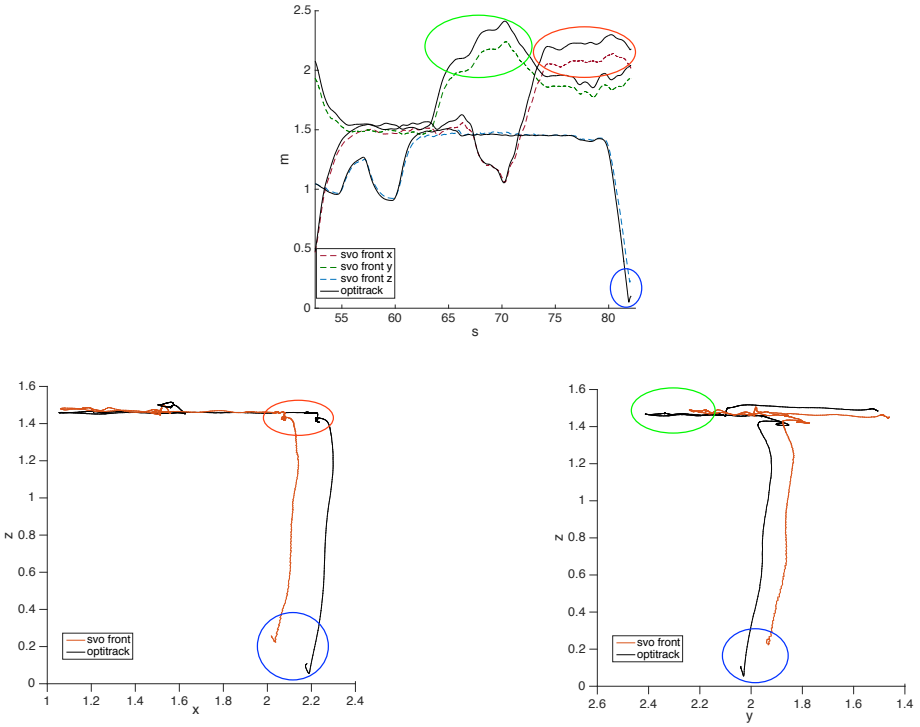


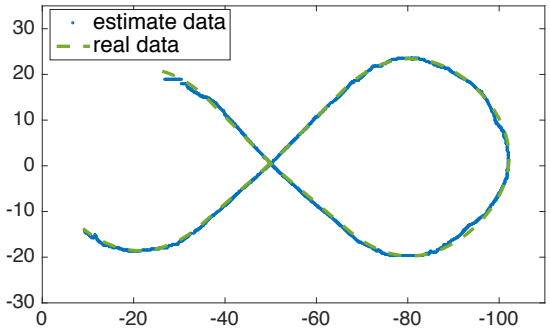
Figure 6.10: Manual flight in which SVO drifted away from the reality. Highlighted by circles are the moment in which these drifts happened and there are the correspondences between the time domain and the trajectory.

6.4 Base detection and tracking

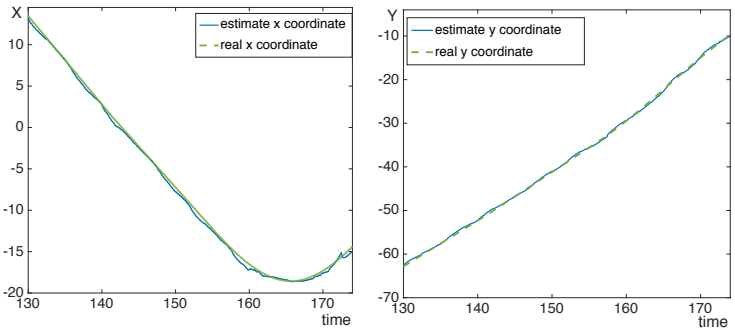
Several experiments were performed both in simulation and in the real world to evaluate the performance of the state estimation.

6.4.1 From high altitude

We are not requiring the state estimation from high altitude to be very precise, we need a rough estimation of the position of the platform, but it is important to be consistent and without lots of jumps.
The precision of the estimation is depending on the altitude from which the quadrotor is tracking the base, but anyway, with the designed EKF we obtain a reliable estimation of the base state.
The following figures show the result of different experiments computed in simulation.



(a) Trajectory



(b) x (c) y

Figure 6.11: Comparison between estimate position (blue dots) and real position (green line) in simulation. The platform is moving in the 8 shape path at 1.5m/s and the quadrotor explores the area at 15m of altitude.

As one can see from the graph 6.12, the estimation error can be really high during this phase. In particular the chart shows the average error between x and y directions and the correspondent RMSE from a searching at 15m of altitude.

This data even if are not really precise are good enough to perform the first stage two and three of the state machine.

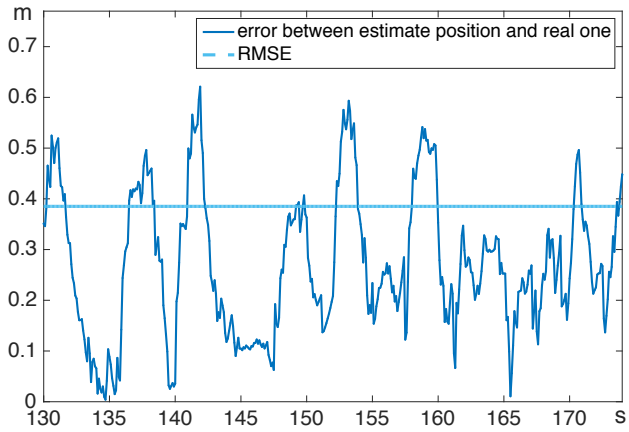


Figure 6.12: Average error between estimate and real x,y coordinate. The RMSE is below 40cm. The value seems high but the estimation is taken from 15m high and while both quadrotor and platform are moving. This precision is sufficient to estimate the type of movement of the base.

TODO images from real world moving platform HIGH ALTITUDE

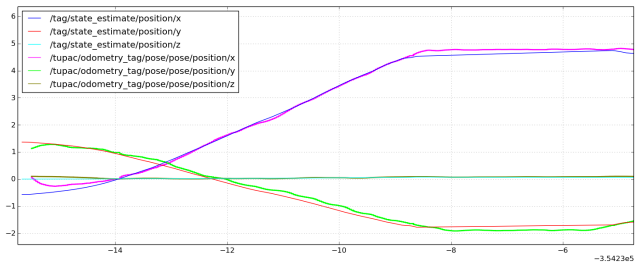


Figure 6.13: Real world test. Comparison between estimate position and ground truth for a platform moving at $1\frac{m}{s}$

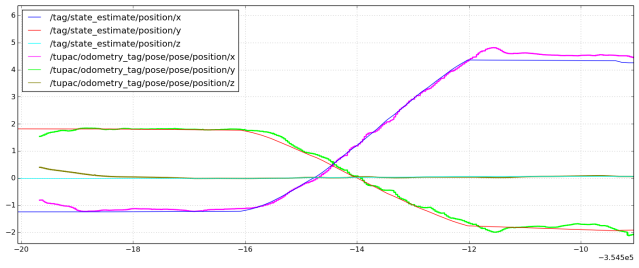


Figure 6.14: Real world test. Comparison between estimate position and ground truth for a platform moving at $2\frac{m}{s}$

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 [40] that uses the AprilTags library [41], AR-Sys [42] and AR-Track-Alvar [43].
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:

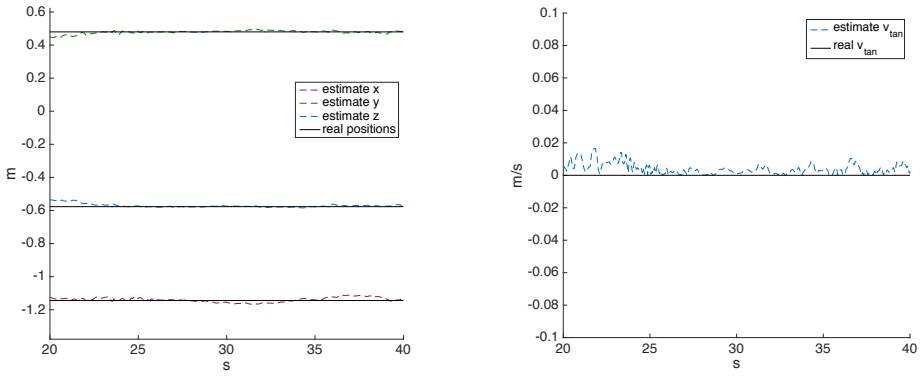
- **Light conditions**: all these methods uses the edge based approach, so the results is similar in different light conditions.
- **Final pose**: all trackers solve a PnP Problem to find the 6dof pose of the camera that minimize the reprojection error of the points on the image. The final result is the transformation between tag and 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 possess the ability to directly track multiple tags or single target composed by multiple tags.
- **Precision**: we measure the error at $1m$ 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: $1Hz$
  - AR-Sys: $4Hz$
  - AR-Track-Alvar: $1Hz$

AR-Sys In our final implementation we decided to use AR-Sys because its computational efficiency. AR-Sys is 3D pose estimation ROS package that uses ARUco marker boards [44].
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.

Results

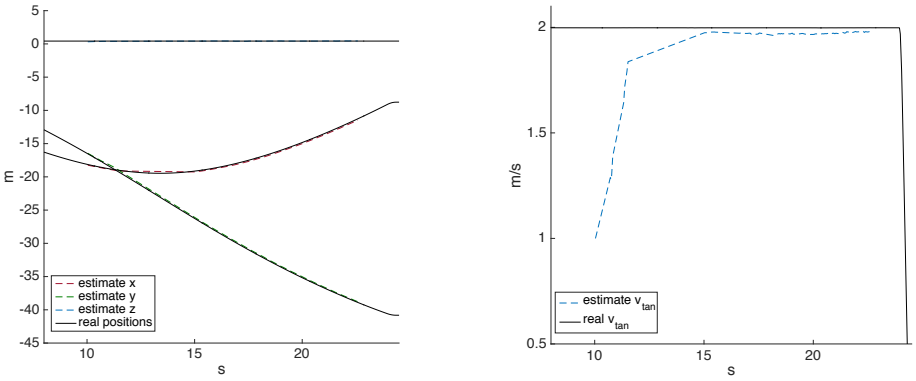
Very accurate pose estimation is obtain when the AR tags are used. Generally the error in the x, y coordinate is less then $10cm$ while in the z direction is about $3cm$.

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



(a) Position (b) Velocity

Figure 6.15: 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 $5cm$ in x, y and $2.5cm$ in z.



(a) Position (b) Velocity

Figure 6.16: Simulation test. Comparison between estimate position and velocity with the ground truth values. The velocity is initialized with a wrong value of 1m/s but the filter needs few steps to converge to the right value of 2m/s. The estimate position has a RMSE of $10cm$ in x, y and $2cm$ in z.

TODO images from real world moving platform

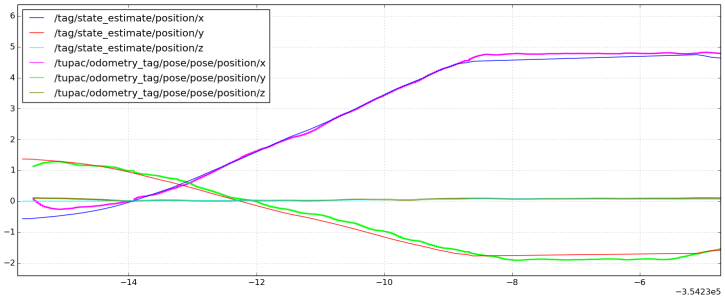


Figure 6.17: Real world test. Comparison between estimate position and ground truth for a platform moving at $1\frac{m}{s}$

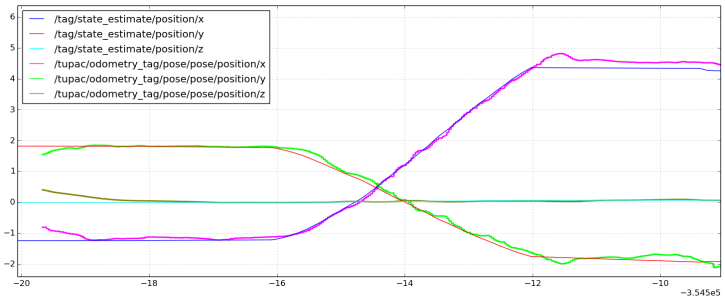


Figure 6.18: Real world test. Comparison between estimate position and ground truth for a platform moving at $2\frac{m}{s}$

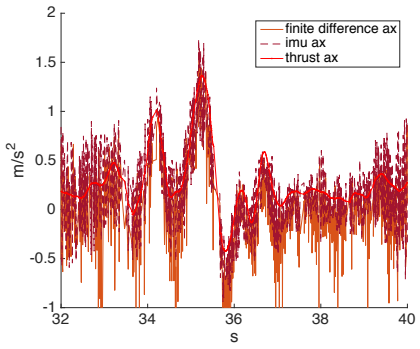
6.5 Trajectory generation

6.5.1 Acceleration estimation

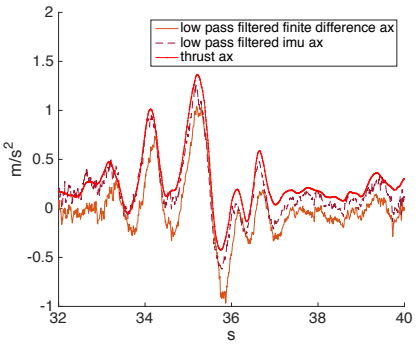
We described in 5.1.6 the three possible methods to estimate the acceleration of the quadrotor at a given time. This calculation is necessary because the trajectory generator needs a full state description of the initial condition of the quadtotor [position,velocity,acceleration] in order to solve the optimal control problem.
We took several data sets while the quadrotor was flying in the flyingroom in order to compare the different methods used to approximate the accelerations.

To understand how the raw data looks like, the figure 6.19 on the first column compares the data taken directly from the IMU, with finite difference calculate with two subsequent velocity estimations (without filtering) and the estimation computed with the total thrust. While on the right column there is 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.

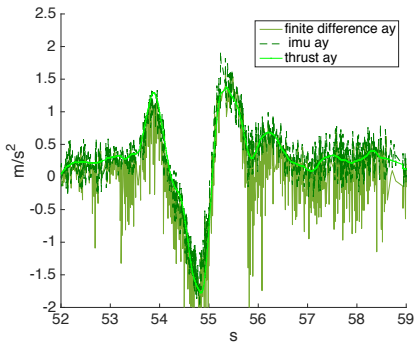
It is easy to notice that only the raw data from the thrust can be considered a good approximation of the acceleration, while the other two signals needs some



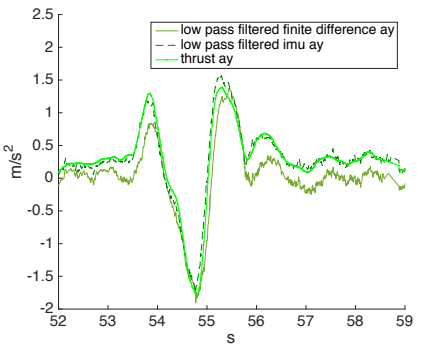
(a) Acceleration x axis not filtered



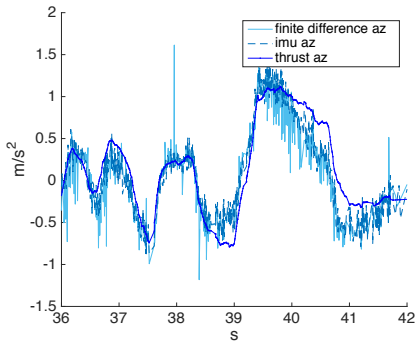
(b) Acceleration x axis filtered



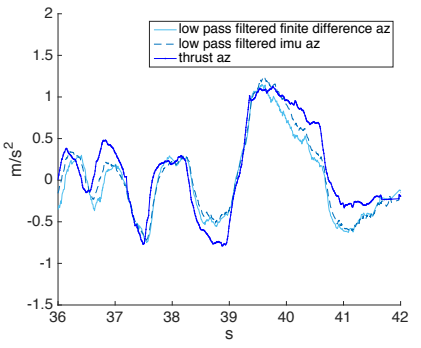
(c) Acceleration y axis not filtered



(d) Acceleration y axis filtered



(e) Acceleration z axis not filtered

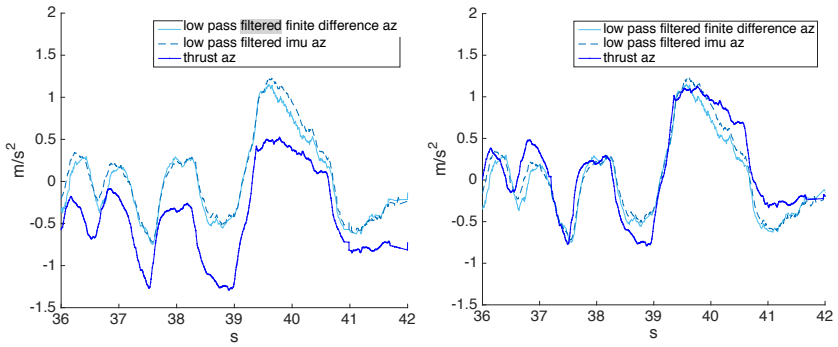


(f) Acceleration z axis filtered

Figure 6.19: 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 data set, but with filtered version for IMU and finite difference.

filtering.
On the other side, the comparison of the filtered data has much more sense. In this case the data from IMU and finite difference are more smooth and can be used, but filtering the high frequencies we slow down the response of these signal and so cause a delay.

In the figure 6.19 we can also notice that there is some offset between the three different approximations considering the same image, in particular in the z directions this difference it is very clear between the acceleration computed with the thrust and the other two methods.
This offset is more evident in the figure 6.20 where the right graph is the same figure 6.19f, while the left part is calculate with the same data set, but the mass of the quadrotor is modify by the 5% from $515g$ to $545g$. This tiny modification is creating a big difference in the final acceleration.



(a) Acceleration z axis filtered with real mass of 545g

(b) Acceleration z axis filtered with modify mass of 515g

Figure 6.20: 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 signal.
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.
We have some videos in which we run the whole system from start to end with successful land on a moving platform.

In simulation we reached velocity of the moving platform up to $2\frac{m}{s}$, with landing

rate of over 90%. While in the real world we managed to achieve velocity of $1\frac{m}{s}$ for the platform. The main issues with higher velocity in the real world is that the state estimation starts to drift and the quadrotor we are using has hardware limitations that do not permit fast trajectory, 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. TODO

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 therms 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 7.2.2).
Following we describe what solution can be applied in future to solve these problems.

7.2.1 State estimation using also GPS and Teraranger

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 continue 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 [45]: 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.

7.2.2 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 response at little variation can solve this problem.
  A method to increase the sensitivity is tuning the controller gains, but this can lead to a unstable behavior, so further studies must be done.
- change both the trajectory and the contorller: implementing a new contorller like a LQR controller [46] that takes in account both the dynamic 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.

7.2.3 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 $k90^o$, 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 Mahony. 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] 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.

[8] 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.

[9] 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.

[10] Daniel Mellinger, Michael Shomin, and Vijay Kumar. Control of quadrotors for robust perching and landing. pages 205–225, 2010.

[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] Athanasios Sideris and James E Bobrow. An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems. *IEEE transactions on automatic control*, 50(12):2043–2047, 2005.

[15] Moses Bangura and Robert Mahony. Real-time model predictive control for quadrotors. *IFAC Proceedings Volumes*, 47(3):11773–11780, 2014.

[16] Mark W Mueller and Raffaello D'Andrea. A model predictive controller for quadrocopter state interception. pages 1383–1389, 2013.

[17] 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.

[18] MBZIRC. Mbzirc challenge description, 2016.

[19] 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.

[20] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. pages 15–22, 2014.

[21] 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.

[22] 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.

[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] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. pages 85–94, 1999.

[30] Mark Fiala. Designing highly reliable fiducial markers. *IEEE Transactions on Pattern analysis and machine intelligence*, 32(7):1317–1324, 2010.

[31] G. Bradski. Opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[32] Serge Belongie. Rodrigues rotation formula. *MathWorld A Wolfram Web Resource*, 1999.

[33] Michael J Van Nieuwstadt and Richard M Murray. Real time trajectory generation for differentially flat systems. 1997.

[34] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. pages 2520–2525, 2011.

[35] Clearpath Robotics. Clearpath robotics, 2009.

[36] Dave Coleman John Hsu, Nate Koenig. Gazebo simulator, 2002.

[37] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Robot operating system (ros): The complete reference (volume 1). pages 595–625, 2016.

[38] Clearpath Robotics. Robots husky ros package.

[39] Optitrack motion caption system.

[40] RPG. Rpg-april-tags ros package, 2014.

[41] Jeffrey Boyland Edwin Olson, Michael Kaess David Touretzky, and Hordur Johannson. April-tags library, 2012.

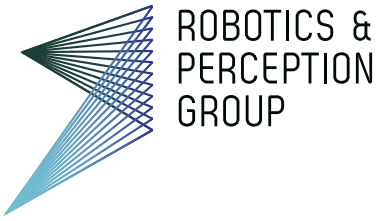
[42] Salinas Bence Magyar Hamdi Sahloul, Rafael Munoz. Ar-sys ros package, 2014.

[43] Scott Niekum. Ar-track-alvar ros package, 2012.

[44] 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.

[45] Tera bee. Teraranger sensor.

[46] Wikipedia. Linear-quadratic regulator, 2016.



Title of work:

Autonomous landing on a moving platform

Thesis type and date:

Master Thesis, September 2016

Supervision:

First Supervisor Davide Falanga
Second Supervisor
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`

Zurich, 20. 9. 2016: _____