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

Compress the introduction in a few key sentences. No more than half a page. The abstract should motivate your work, outline the work that you did, and give some insights into its results.

Nomenclature

Notation

$\mathbf{J}$	Jacobian
$\mathbf{H}$	Hessian
$\mathbf{T}_{WB}$	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

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
DoF	Degree of Freedom
IMU	Inertial Measurement Unit
MAV	Micro Aerial Vehicle
ROS	Robot Operating System

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 increasingly difficult settings. For many of those scenarios is necessary autonomous landing of the UAV on a small base using only on- board sensors. As a matter of fact, major drawback of current civil Micro Air Vehicles (MAVs) is the limited flight time. Automated landing systems, along with suitable recharging platforms, enable longer term UAV missions with greater autonomy. Furthermore there are several applications in which the landing target is not static, but it is moving w.r.t the world coordinate frame, so the quadrotor must be able to perform a precise landing maneuver over this moving platform, this situation would occur, for example, when landing on a moving ship or car.

Highly accurate localization is required in order to allow the MAV to land precisely over the platform. Most of UAVs are provided by 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 on- board cameras: vision based approaches, to state estimation, 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

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

the ground station, which allows great processing power, but leads to restrictions in autonomy on the UAV.

At the beginning the research was about landing on a static platform. Hardware and tequinques used to achive the succesful completement 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.

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 measurement noise. Or Herisse in [4] where only optical flow is used for hovering flight and vertical landing control.

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 on landing on a moving platform.

Wenzel in [7] is performing the following 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 result. The moving platform is not fast, $0.4m/s$, and is moving both in a circular path or emulating a ship turning on water.

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 $0.7m/s$ and is moving always in a straight line. When we estimate the state of the moving base, the knowledge of the type of moment of the platform, can be crucial in order to filter noisy measures and predict where the target will be within t seconds.
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. 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 landing base state estimation, and his algorithm consists in a precomputed trajectory followed by a position-altitude control based on the linearized system of the quadrotor.
An interesting part 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 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, body-rates) 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 related works is that to design a landing framework we need:

- a good estimation of the UAV's state and platform's state.
- a "manager" that considering the state estimations define the current phase in which the system is.
- a MPC-like algorithm that increases the robustness updating continuously the future actions that must be applied to the UAV.

Several papers have been written on MPC [12] applied to control of the quadrotor.

MPC Fast Nonlinear MPC for unified trajectory optimization and tracking + An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems (algorithm SQL is explained) Generate optimal feedforward and feedback control gains Resolve unconstrained MPC problem with SLQ algorithm repeatedly Closed loop that applies gains calculated previous point Cost

function with desired state-input ??? (no trajectory I do not understand what they should be, only last is final point) and waypoints Predict the system-¿ understand if there are time lag between when a policy is calculated and when it is applied Initialization SQL or with previous solution or LQR controller H (final state weight) is solution of infinite horizon LQR Ricatti equation

Real-time MPC for quadrotors Contorol Low-level control motors Mid-level control attitude -¿ HIGH gain control to track accurately reference angular velocity (dynamic reduction) dynamically extend the thrust command ??? High-level control trajectory tracking -¿ Feedback linearization to create a system handleable with MPC that creates input that will map in input for mid-level control MPC in the feedback equivalent system -¿ input applied on the original system ONBOARD MPC gives a desired angular velocity, from this we can calculate the derided thrust and then calculate the feedback torque (eq 12) Uses desired angular velocity as input

A MPC for quadrocopter state interception Generate trajectories from initial and final state Formulating trajectory in jerk and then solving convex optimization problem on each decoupled axis Feasibility constraint of input Cost functions with jerks is kind of minimizing the product of the inputs Using FORCES to solve the optimization problem

A computationally efficient Motion primitive for quadrocopter trajectory generator.

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 on-board.

Chapter 2

MBZIRC challenge

The Mohamed Bin Zayed International Robotics Challenge (MBZIRC) is an international robotics competition, held every two years [13]. MBZIRC provides an ambitious and technologically demanding set of challenges, that aim to inspire the future of robotics through innovative solutions and technological excellence.
Robotics is poised to have a transformative impact in a variety of new markets and on various human social aspects. These include robot applications in disaster response, oil and gas, manufacturing, construction and household chores. Enabling technologies for such applications include robots working more autonomously in dynamic, unstructured environments, while collaborating and interacting with other robots and humans.
The competition consists in 3 challenges and this thesis consist in developing a framework to complete challenge number 1 that requires an UAV to land 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.

2.0.1 The Arena

The challenge will be performed in an arena with the following specifications (see figure 2.1):

- 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.

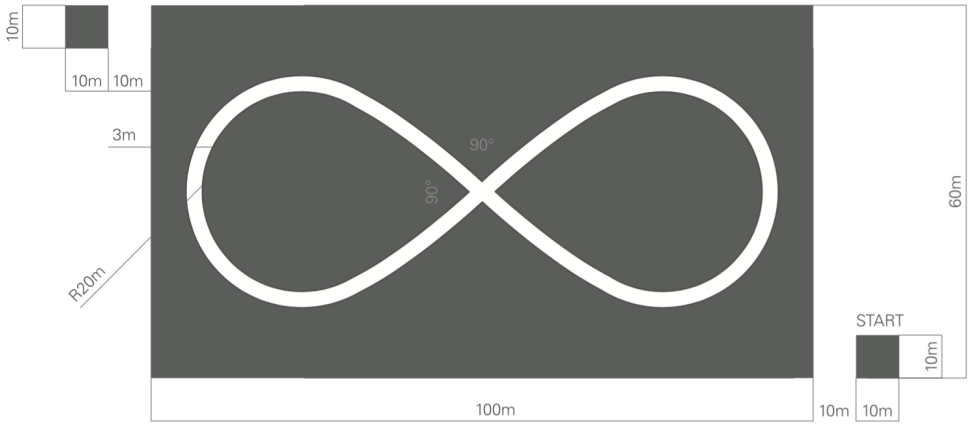


Figure 2.1: Arena of the challenge

2.0.2 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 will be 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 2.2. A successful landing is when a point of contact of the UAV is within the landing circle, with propulsion off and rotors not spinning.

2.0.3 Infinity shape path

The moving platform will move in an infinity-shape path described in the figure 2.1. 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 r_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.

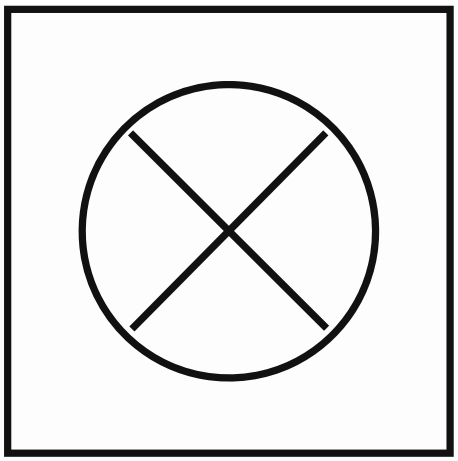


Figure 2.2: Design of the platform in which the quadrotor must land on

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:

$$\begin{aligned} y &= x \\ y &= -x \end{aligned} \tag{2.1}$$

while the two circles

$$\begin{aligned} y^2 + (x - x_0)^2 &= r_8^2 \\ y^2 + (x + x_0)^2 &= r_8^2 \end{aligned} \tag{2.2}$$

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

$$x_0 = \frac{\sqrt{2}}{2} r_8 \tag{2.3}$$

That correspond to the 4 intersections coordinate

$$\Big(\frac{\sqrt{2}}{2} r_8, \frac{\sqrt{2}}{2} r_8\Big); \Big(\frac{\sqrt{2}}{2} r_8, -\frac{\sqrt{2}}{2} r_8\Big); \Big(-\frac{\sqrt{2}}{2} r_8, -\frac{\sqrt{2}}{2} r_8\Big); \Big(-\frac{\sqrt{2}}{2} r_8, \frac{\sqrt{2}}{2} r_8\Big) \tag{2.4}$$

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:

$$\theta(x) = \begin{cases} -\frac{x}{r_8} & x \in \Big[0, \frac{3\pi}{4} r_8\Big] \\ -\frac{3\pi}{4} & x \in \Big[\frac{3\pi}{4} r_8, \frac{3\pi}{4} r_8 + r_8\Big] \end{cases} \tag{2.5}$$

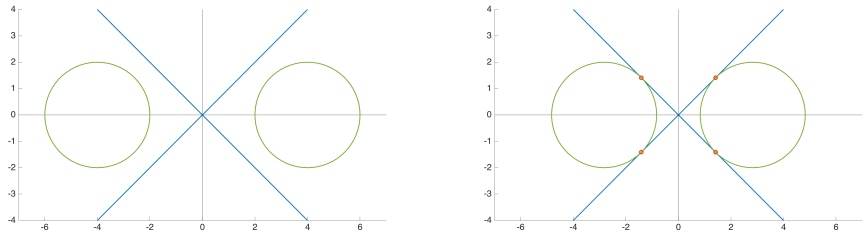


Figure 2.3: How to construct the infinity-shape path

This function define a quarter of the trajectory 2.4 in function of the radius r_8 of the path.
It is now possible to use it to generate the entire trajectory $(x(t), y(t))$ 2.5:
we know that the length of the path is

$$l = 4(\frac{3\pi}{4}r_8 + r_8)$$

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:

$$\begin{aligned} \dot{x} &= v_{tan} cos(\theta(t)) \\ \dot{y} &= v_{tan} sin(\theta(t)) \end{aligned} \tag{2.6}$$

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

$$\begin{aligned} x_k &= x_{k-1} + dt\big(v_{tank-1} cos(\theta_{k-1})\big) \\ y_k &= y_{k-1} + dt\big(v_{tank-1} sin(\theta_{k-1})\big) \end{aligned} \tag{2.7}$$

2.1 Simulation

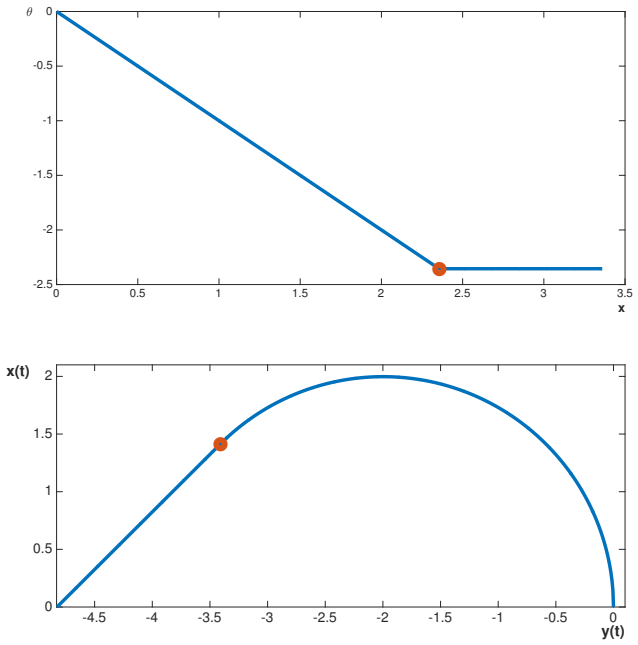


Figure 2.4: The parametrization of a quarter of the path

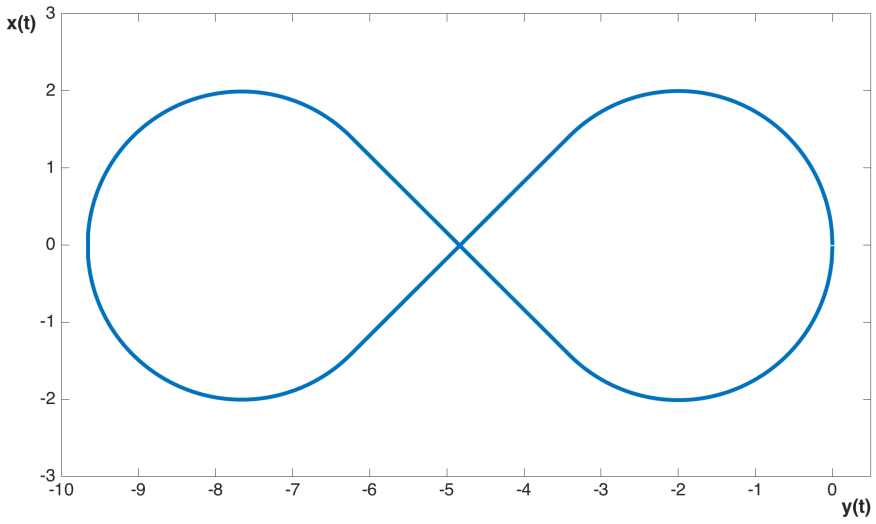


Figure 2.5: Infinity-shape path

Chapter 3

General Framework

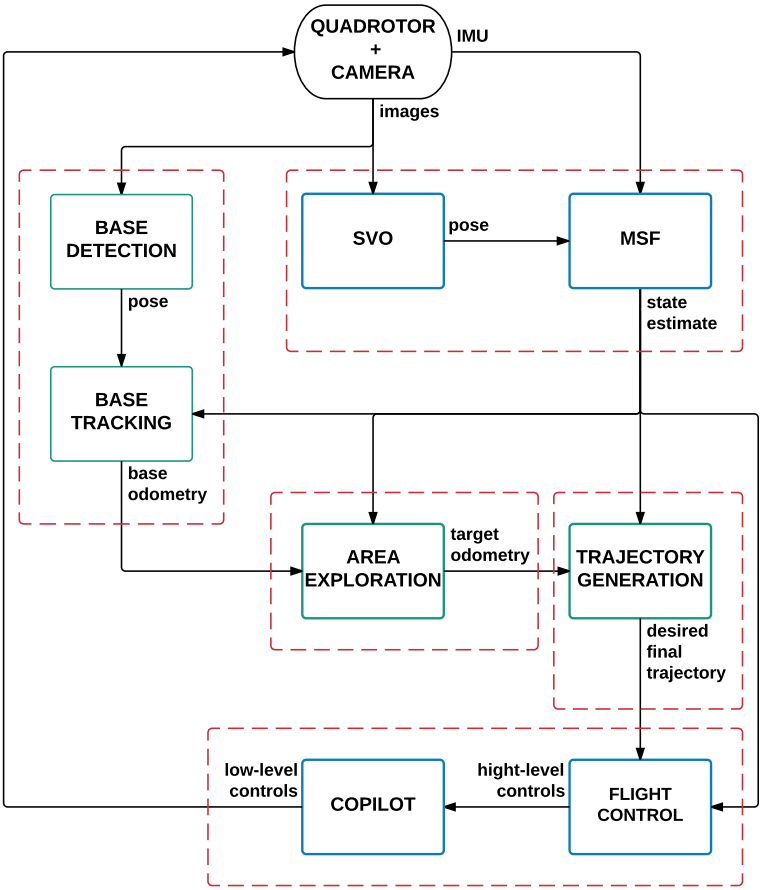


Figure 3.1: Pipeline

3.0.1 SVO & MSF

are calculating the best estimation of the state of quadrotor based on data from a camera and an IMU

SVO front looking camera

explanation about base in fov
Fish eye or not fish eye? pros and cons

3.0.2 Base Detection & Tracking

given images taken from a camera on the quadrotor, it is searching the landing platform and estimating its state

3.0.3 Area Exploration

3.0.4 Trajectory Generation

considering the state of the quadrotor and of the landing platform, they are calculating where the quadrotor must go and with which trajectory.

3.0.5 Flight Control & Copilot

given the desired trajectory, they are computing the controls that must be applied to the quad in order to follow such trajectory

Chapter 4

Base Detection and Tracking

One part of the work is focused on the estimation of the state 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 and where the quadrotor should go to successfully complete the mission.

An Extended Kalman Filter [14] is design in order to have the most reliable value of the state of the platform.
Kalman filtering is an algorithm that uses a series of noisy measurements observed over time and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone, by 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 Kalman filter produces estimates of the current state variables, along with their uncertainties, based on a model of the system:

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{4.1}$$

- Once the outcome of the next measurement is observed:

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k \tag{4.2}$$

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

In the extended Kalman filter, the state transition and observation models don't need to be linear functions of the state but may instead be differentiable functions.

($\boldsymbol{w}_k$ 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. $\boldsymbol{u}_k$ is the control vector).

The algorithm is recursive. 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 Kalman filter 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.
Initialization

$$\begin{aligned} \boldsymbol{x}_{0|0} &= x_0 \\ \boldsymbol{P}_{0|0} &= P_0 \end{aligned} \tag{4.3}$$

In this case the prediction equations are continuous in time, so for the prediction step of the EKF we have to solve:

$$\begin{aligned} \dot{\hat{\boldsymbol{x}}}(t) &= f(\hat{\boldsymbol{x}}(t), \boldsymbol{u}(t)) \\ \dot{\boldsymbol{P}}(t) &= \boldsymbol{F}(t)\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{F}(t)^\top + \boldsymbol{Q}(t) \end{aligned} \tag{4.4}$$

for $t \in (t_{k-1}, t_k)$ where

$$\begin{aligned} \hat{\boldsymbol{x}}(t_{k-1}) &= \hat{x}_{k-1|k-1} \\ \boldsymbol{P}(t_{k-1}) &= P_{k-1|k-1} \\ \boldsymbol{F}(t) &= \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}(t), \boldsymbol{u}(t)} \\ \hat{\boldsymbol{x}}_{k|k-1} &= \hat{\boldsymbol{x}}(t_k) \\ \boldsymbol{P}_{k|k-1} &= \boldsymbol{P}(t_k) \end{aligned} \tag{4.5}$$

In order to save some computation we can discretize the dynamicin order to have shorter computation during the prediction step of the EKF:

$$\begin{aligned} \hat{\boldsymbol{x}}_{k|k-1} &= f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k) \\ \boldsymbol{P}_{k|k-1} &= \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^\top + \boldsymbol{Q}_k \end{aligned} \tag{4.6}$$

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

$$\boldsymbol{F}_{k-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k} \tag{4.7}$$

While the update equations are discrete in time and they yield to the following update step:

$$\begin{aligned} \boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^\top(\boldsymbol{H}_k\boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^\top + \boldsymbol{R}_k)^{-1} \\ \hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1})) \\ \boldsymbol{P}_{k|k} &= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k|k-1} \end{aligned} \tag{4.8}$$

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

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{4.9}$$

4.1 Prediction update: non-holonomic model

The platform is considered as a car and simulated with a non-holonomic model 4.1. In this model the state is defined as $\boldsymbol{x} = (x, y, z, \theta, v, \phi)$: It corresponds to

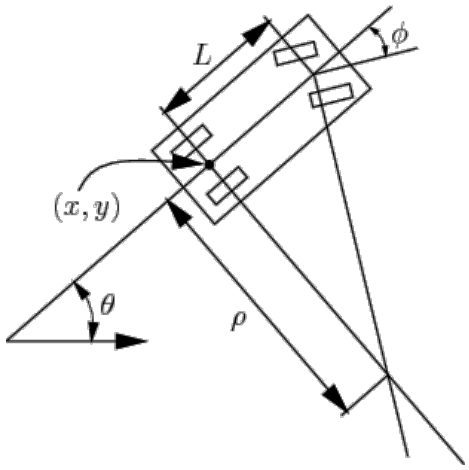


Figure 4.1: Non-holonomic model

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) 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 v and in the angle of curvature ϕ.
The equation of motion in continuous time are:

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

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \\ \dot{v} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} vcos(\theta) \\ vsin(\theta) \\ 0 \\ \frac{v}{L} tan(\phi) \\ u_1 \\ u_2 \end{bmatrix} \tag{4.10}$$

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

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

with $\boldsymbol{x}_k = \boldsymbol{x}(t_k)$, $\boldsymbol{x}_{k-1} = \boldsymbol{x}(t_{k-1})$, $dt = t_k - t_{k-1}$

$$\boldsymbol{x_k} = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_k \\ v_k \\ \phi_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + dt\big(v_{k-1}cos(\theta_{k-1})\big) \\ y_{k-1} + dt\big(v_{k-1}sin(\theta_{k-1})\big) \\ z_{k-1} \\ \theta_{k-1} + dt\Big(\frac{v_{k-1}}{L}tan(\phi_{k-1})\Big) \\ v_{k-1} + dt\big(u_{1k}\big) \\ \phi_{k-1} + dt\big(u_{2k}\big) \end{bmatrix} \tag{4.11}$$

In order to solve the former system, we have anyway to find a numerical solution. For this purpose we use a Runge-Kutta scheme [15].
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:

$$\begin{aligned} \dot{y} &= f(t, y) \\ y(t_0) &= y_0 \end{aligned} \tag{4.12}$$

y is an unknown function (scalar or vector) of time t, which we would like to approximate and the function f and the data t_0, y_0 are given.
Now pick a step-size $h > 0$ and define

$$\begin{aligned} y_{k+1} &= y_k + \tfrac{h}{6}\left(\alpha_1 + 2\alpha_2 + 2\alpha_3 + \alpha_4\right), \\ t_{k+1} &= t_k + h \end{aligned} \tag{4.13}$$

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

$$\begin{aligned} \alpha_1 &= f(t_k, y_k), \\ \alpha_2 &= f(t_k + \frac{h}{2}, y_k + \frac{h}{2}\alpha_1), \\ \alpha_3 &= f(t_k + \frac{h}{2}, y_k + \frac{h}{2}\alpha_2), \\ \alpha_4 &= f(t_k + h, y_k + h\alpha_3). \end{aligned} \tag{4.14}$$

Here y_{k+1} is the RK4 approximation of $y(t_{k+1})$, and it is determined by the present value (y_k) plus the weighted average of four increments.

Straight and circular path

We assume the input u_1 and u_2 are equal to zero. In this way we can have three type of movement:

- the platform can be static ($v_f(0) = 0$)

- it can move in a straight line ($v_f(0) \neq 0$ and $\phi(0) = 0$)
- it can move in a circle ($v_f(0) \neq 0$ and $\phi(0) \neq 0$)

As described in charapter , 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 this model can be used also in the final implementation.

4.2 Measurement update

From equation 4.10 we have the variables that describe the state of the moving car. We have to be able to measure some of these components in order to 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 the relative position and orientation. At this point knowing the position of the camera in the real world we can measure the following vector $\boldsymbol{z} = (x, y, z, \theta)$. 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 measurement equations are simply:

$$\boldsymbol{z_k} = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_k \end{bmatrix} \tag{4.15}$$

In the different phases we have to use different methods measure $\boldsymbol{z}$ detecting and tracking the base:

- 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 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 a theoretical model of the movement.

4.2.1 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 from the down

looking camera to find a moving white square and calculate its optical flow to predict its future position.

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 [16] 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:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{4.16}$$

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

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t. \tag{4.17}$$

From these equations it follows that:

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0 \tag{4.18}$$

which results in

$$I_x V_x + I_y V_y + I_t = 0 \tag{4.19}$$

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:

$$\nabla I^T \cdot \vec{V} = -I_t \tag{4.20}$$

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 [17]. 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:

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n) \end{aligned} \tag{4.21}$$

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 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

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix} \tag{4.22}$$

This system has more equations than unknowns and thus it is usually over-determined. The Lucas-Kanade method obtains a compromise solution by the least squares principle. Namely, it solves the 2×2 system:

$$\begin{aligned} A^T A v &= A^T b \\ \text{v} &= (A^T A)^{-1} A^T b \end{aligned} \tag{4.23}$$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_y(q_i) I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix} \tag{4.24}$$

With this method we can track the interesting points from frame to frame, calculate direction and velocity of the car and so predict where it will be after a time t.

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

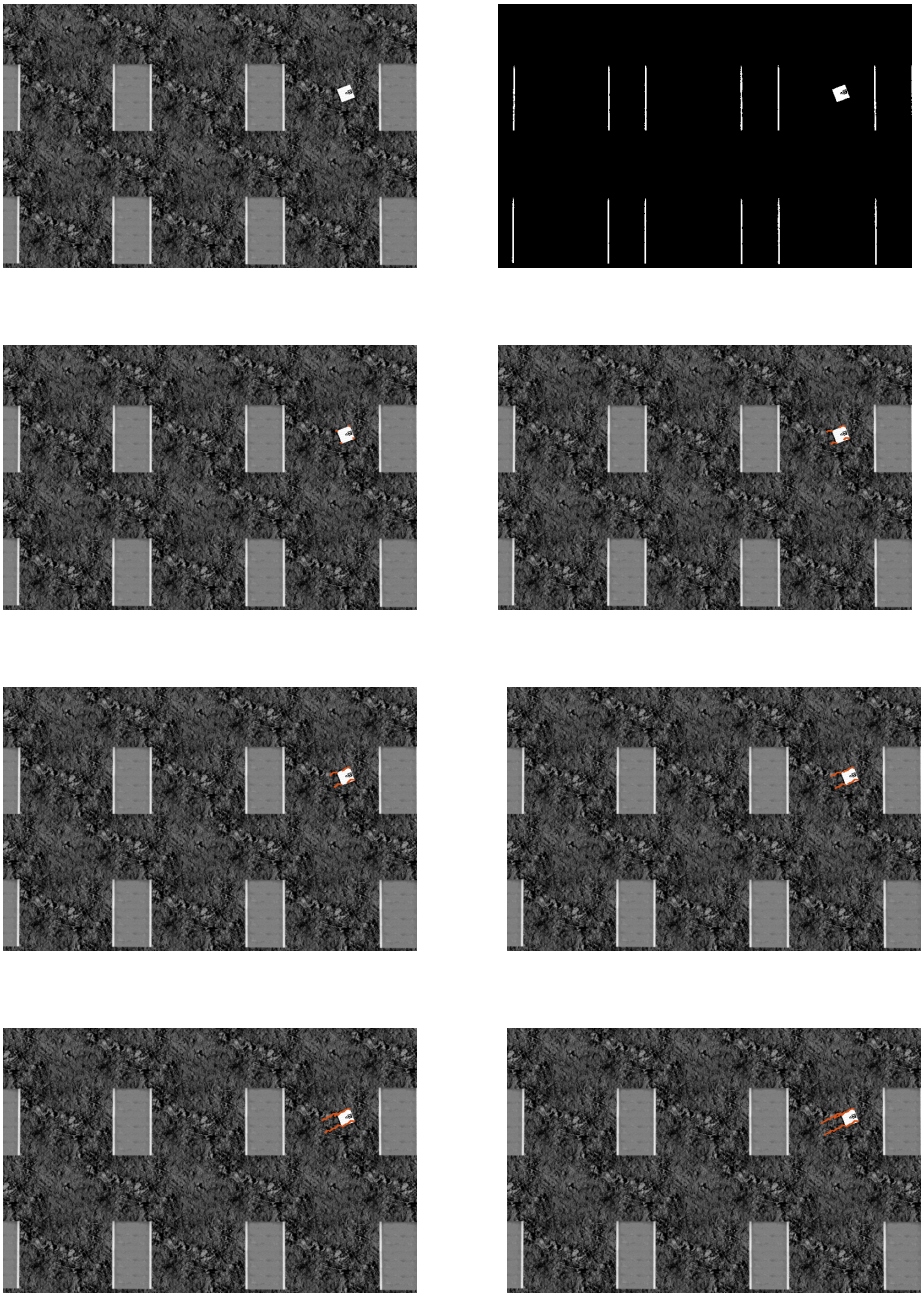


Figure 4.2: A sequence of images where the moving car is detected and tracked. First image is the original image. Then the one after thresholding. Then all the subsequent images where the corners of the platform are tracked.

[18]:

$$wm = A[R|t]M \tag{4.25}$$

In expanded form:

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.26}$$

Where:

- m homogeneous coordinate of the projection point in pixel.
- M homogeneous coordinate of a 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. 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

$$z = \frac{l_w f}{l_i} \tag{4.27}$$

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 4.3:

- 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 $\sqrt{2}l_i$

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.

If this depth $z! = 0$ we can solve the system of equation 4.25 finding an unique

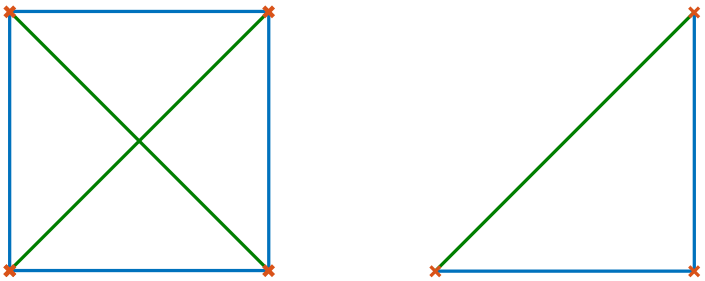


Figure 4.3: Model of the square platform detected on the image. Red crosses corner detected. Blue lines edges with length l_i. Green lines edges with length $\sqrt{2}l_i$

solution using the following equivalent equations:

$$\begin{aligned} x &= z\frac{u - c_x}{f_x} \\ y &= z\frac{v - c_y}{f_y} \\ \begin{bmatrix} x \\ y \\ z \end{bmatrix} &= R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \end{aligned} \tag{4.28}$$

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 [19] 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. The only 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, but when all the 4 points are available we solve the correspondent PnP problem to find a better estimation of the base position.

4.2.2 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 2.2, while for the first testing another design is considered 4.4, in order to use preexisting algorithm that allow pose estimation with respect to the camera. 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 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 [20]), 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 [21]), 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, the four corners of every potential marker are used to calculate a homography in order to remove the perspective distortion. This perspective undistortion is done solving a Perspective-n-Point problem [19].
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.

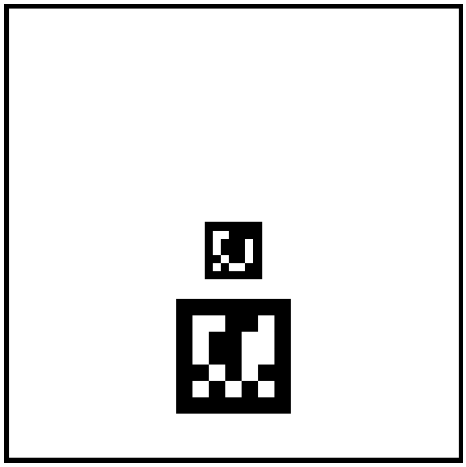


Figure 4.4: Design of the platform, we are using, in which the quadrotor must land on

Implementation

In the real world implementation we tried several different tag detector ROS packages, such as RPG-April-Tags [22] that uses the AprilTags library [23], AR-Sys [24] and AR-Track-Alvar [25]. All of them have some strengths and weaknesses:

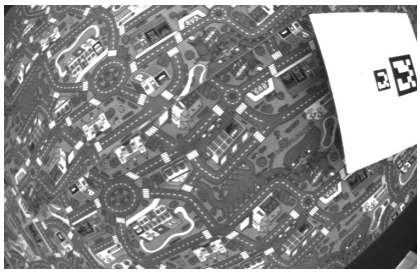
- Light conditions: all these methods uses the edge based approach, so the results is similar in different light conditions.

- 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 objects and object composed by multiple tags.
- Precision:we measure the error at $1m$ distance from the tag
  - RPG-April-Tags: $\pm 1pixel$
  - AR-Sys: $\pm 2pixel$
  - AR-Track-Alvar: $\pm 1pixel$
- Frequency: on the quadrotor the performance of the three tracker where quite different
  - RPG-April-Tags: $1Hz$
  - AR-Sys: $4Hz$
  - AR-Track-Alvar: $1Hz$

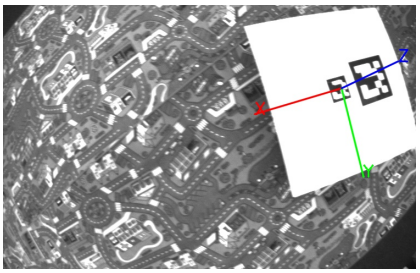
AR-Sys In our final implementation we decided to use AR-Sys because its computational efficiency. AR-Sys is 3D pose estimation ROS package that uses ARUco marker boards [26].
This package guarantees a good error correction in the identification of a specific tag, and, more interesting for our application, a solution to the occlusion problem using multiple markers. This package can identify the pose of boards composed by multiple tags considered as a single unit. This guarantees more stable pose estimates and robustness to occlusions. 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 tag, the pose of the camera is given with respect to this tag.

4.2.3 Covariance Estimation

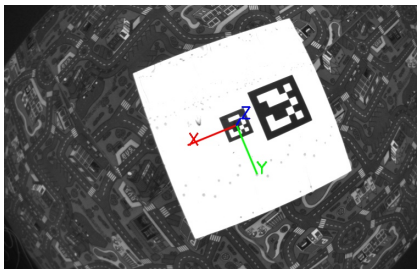
In the practical implementation of the Kalman Filter is crucial to find a good estimate of the noise covariance matrices Q_k and R_k for the prediction and the measurement steps.
When a manual tuning is required, these matrices are considered diagonal, such as each component of the state vector is corrupted 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 correct values for them.
The equations 4.1 is the general matrix formulation of the system corrupted by a multivariable Gaussian noise $\boldsymbol{w}_k$. Now, if we consider the covariance matrix



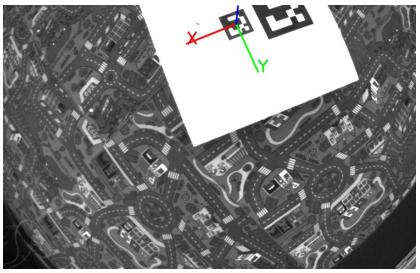
(a) If we are far from the moving platform we have to use the big tag to identify the base.



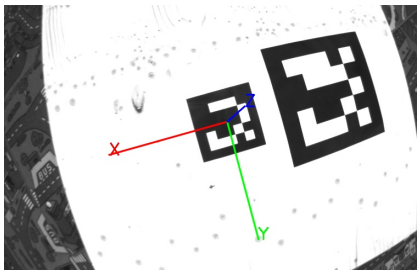
(b) So only when the bigger square is inside the FOV we can detect the center of the base correctly.



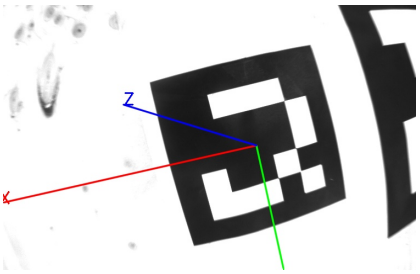
(c) When both the tags are visible we use all the information to have the best position of the master tag.



(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 4.5: 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.

Q_k diagonal we can split the equation 4.1 into:

$$\begin{bmatrix} \dot{x}_k^1 \\ \dot{x}_k^2 \\ \vdots \\ \dot{x}_k^n \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \\ f_2(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \\ \vdots \\ f_n(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \end{bmatrix} + \begin{bmatrix} w_k^1 \\ w_k^2 \\ \vdots \\ w_k^n \end{bmatrix} \tag{4.29}$$

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 4.2 the concept is the same:

$$\begin{bmatrix} z_k^1 \\ z_k^2 \\ \vdots \\ z_k^m \end{bmatrix} = \begin{bmatrix} h_1(\boldsymbol{x}_{k-1}) \\ h_2(\boldsymbol{x}_{k-1}) \\ \vdots \\ h_m(\boldsymbol{x}_{k-1}) \end{bmatrix} + \begin{bmatrix} v_k^1 \\ v_k^2 \\ \vdots \\ v_k^m \end{bmatrix} \tag{4.30}$$

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.
In our case the measures are 4.15 and they are computed with the methods described in the previous section. To estimate the variance of the measure we must start from the error computed in the image when we detect the pose of the platform and propagate the correspondent covariance in the real world. The propagation of uncertainty is the effect of variables' uncertainties on the uncertainty of a function based on them.
Supposed that in the equation 4.2 $\boldsymbol{z}_k$ is not a direct measure of $\boldsymbol{x}_k$ but is a function of other variables $\boldsymbol{\gamma}_k$, such as:

$$h(\boldsymbol{x}_k) = g(\boldsymbol{\gamma}_k) \tag{4.31}$$

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

In the linear case

$$g(\boldsymbol{\gamma}_k) = \mathrm{A}\boldsymbol{\gamma}_k \tag{4.32}$$

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

$$\begin{aligned} Cov(g) &= Cov(\mathrm{A}\boldsymbol{\gamma}_k) = \mathrm{A}Cov(\boldsymbol{\gamma}_k)\mathrm{A}^\top \\ \Sigma_g &= \mathrm{A}\Sigma_{\boldsymbol{\gamma}}\mathrm{A}^\top \end{aligned} \tag{4.33}$$

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:

$$g_i(\boldsymbol{\gamma}_k) \approx g_i(\tilde{\boldsymbol{\gamma}}_k) + \sum_j^n \frac{\partial g_i}{\partial \gamma_j}\Big|_{\tilde{\gamma}_k^j} (\gamma_k^j - \tilde{\gamma}_k^j) \tag{4.34}$$

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:

$$g(\boldsymbol{\gamma}_k) \approx g(\tilde{\boldsymbol{\gamma}}_k) + J\Big|_{\tilde{\gamma}_k} (\gamma_k - \tilde{\gamma}_k) \tag{4.35}$$

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 $A = J$:

$$\Sigma_g \approx \mathrm{J}\Sigma_{\boldsymbol{\gamma}}\mathrm{J}^\top \tag{4.36}$$

In our case to have the final measure $\boldsymbol{z}_k$ in the 3D real world coordinate system, we start from a measure in pixel in the image coordinate system. We have the function $g : \mathrm{R}^{6\times6} \to \mathrm{R}^{2\times2}$ that is converting the real world coordinate in image coordinate, and its Jacobian matrix $J \in \mathrm{R}^{2\times6}$.
Now to calculate the covariance of the final pose estimate, $\Sigma_{RW} \in \mathrm{R}^{6\times6}$, from the covariance of the image position $\Sigma_I \in \mathrm{R}^{2\times2}$, we need to invert the equation 4.37:

$$\Sigma_{RW} = (\mathrm{J}^\top\Sigma_I\mathrm{J})^{-1} \tag{4.37}$$

In our implementation we do not have a single function g from image pixel (u, v) to 3D coordinate (x, y, z, θ). The passage from image to 6dof pose is done using the openCV function *solvePnP* [27] that returns a pose expressed as 3D coordinates (x, y, z) and Rodrigues orientation [28]. To convert the orientation in Euler angles we convert the Rodrigues angles into a rotation matrix and the rotation matrix into finally $(roll, pitch, yaw)$ notation:

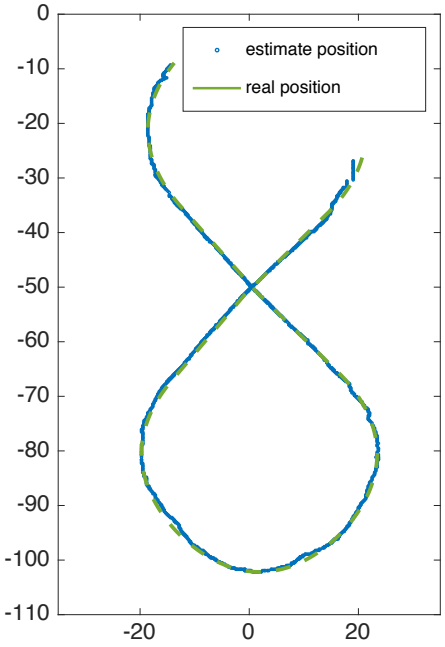
$$\begin{aligned} J_{solvePnP} = J_0 &= \Big[\frac{\partial g}{\partial rodrigues}, \frac{\partial g}{\partial xyzpos}\Big] \\ J_{rodriguesToR} = J_1 &= \frac{\partial rodrigues}{\partial R}, \\ J_{RToEuler} = J_2 &= \frac{\partial R}{\partial euler}, \\ J_{Final} = J &= \Big[\frac{\partial g}{\partial euler}, \frac{\partial g}{\partial xyzpos}\Big] = J_0 \begin{bmatrix} J_1J_2 & 0 \\ 0 & I \end{bmatrix} \end{aligned} \tag{4.38}$$

4.3 Results of the tracking

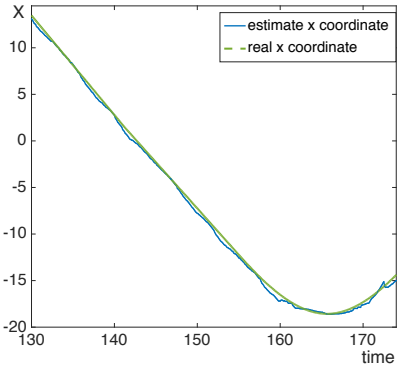
Several experiments were performed both in simulation and in the real world to evaluate the performance of the state estimation.

4.3.1 From high altitude

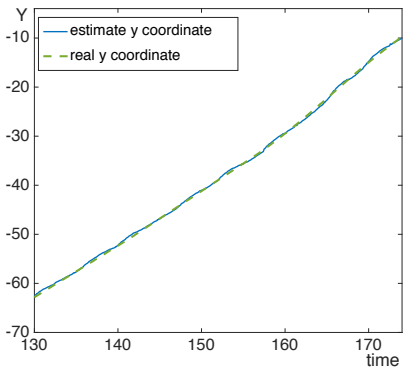
We are not requiring that the state estimation from high altitude is very precise, we need a rough estimation of the position of the platform. With the designed EKF we obtain a reliable estimation with an RMSE error in both x, y coordinate of $40cm$



(a) Trajectory



(b) x



(c) y

Figure 4.6: Comparison

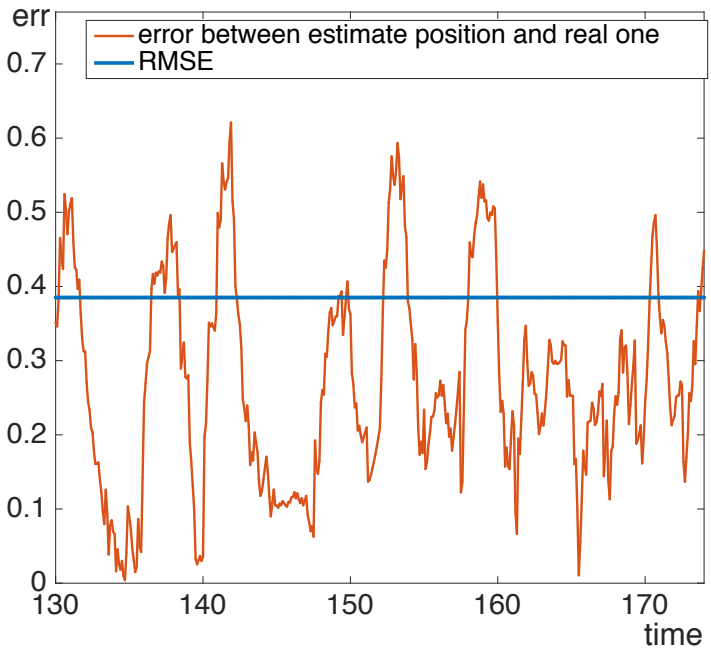
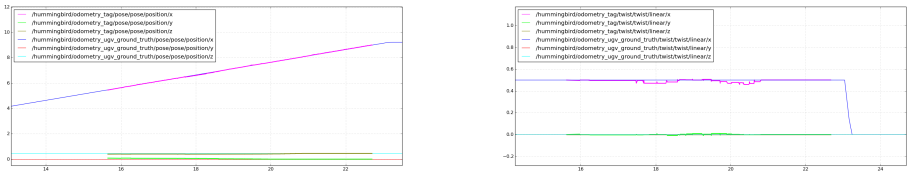


Figure 4.7: error

4.3.2 From low altitude

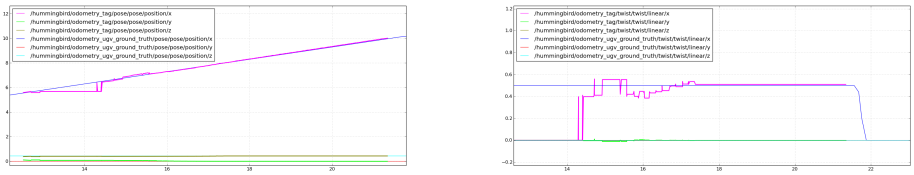
Very accurate pose estimation is obtain when the AR tags are used. Generally the error in the x, y coordinate is less then $5cm$ while in the z direction is about $3cm$.



(a) Position (b) Velocity

Figure 4.8: Simulation test. Comparison between estimate position and velocity with the ground truth values. The velocity is initialized with the right value.

4.4 State Prediction



(a) Position (b) Velocity

Figure 4.9: Simulation test. Comparison between estimate position and velocity with the ground truth values. The velocity is initialized with 0 value. In this case the filter needs some time to converge to the right value of velocity.

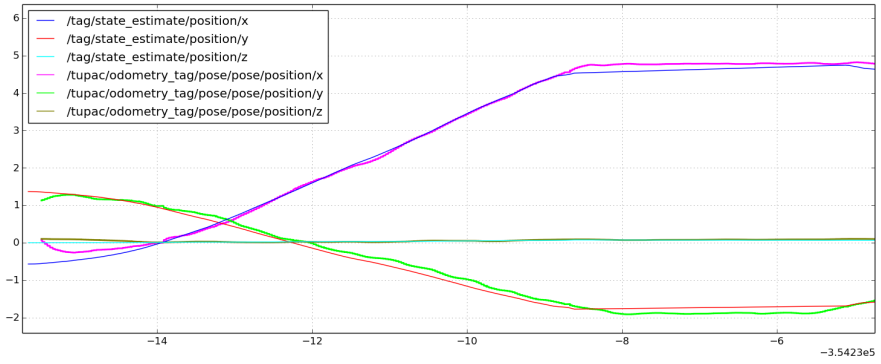


Figure 4.10: Real world test. Comparison between estimate position and ground truth for a platform moving at $1\frac{m}{s}$

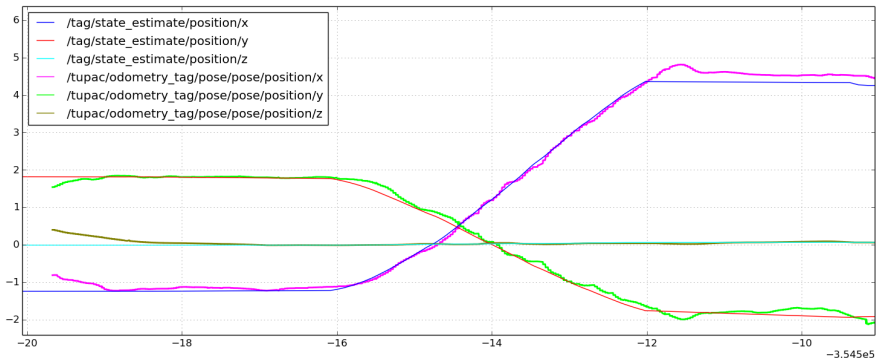


Figure 4.11: Real world test. Comparison between estimate position and ground truth for a platform moving at $2\frac{m}{s}$

Chapter 5

Area exploration

different time exploration imu to switch off time following

jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg
jksdbkvbdsjg

5.1 Phases

5.1.1 First phase - Searching for the base

In this phase the quadrotor starts from a given position and has to find the moving car. Given the rectangle in which the platform can move the UAV follows a list of way-points in order to span the whole area at high altitude. In this way the downward camera can collect information from a large section of the space and the searching task can be performed faster.

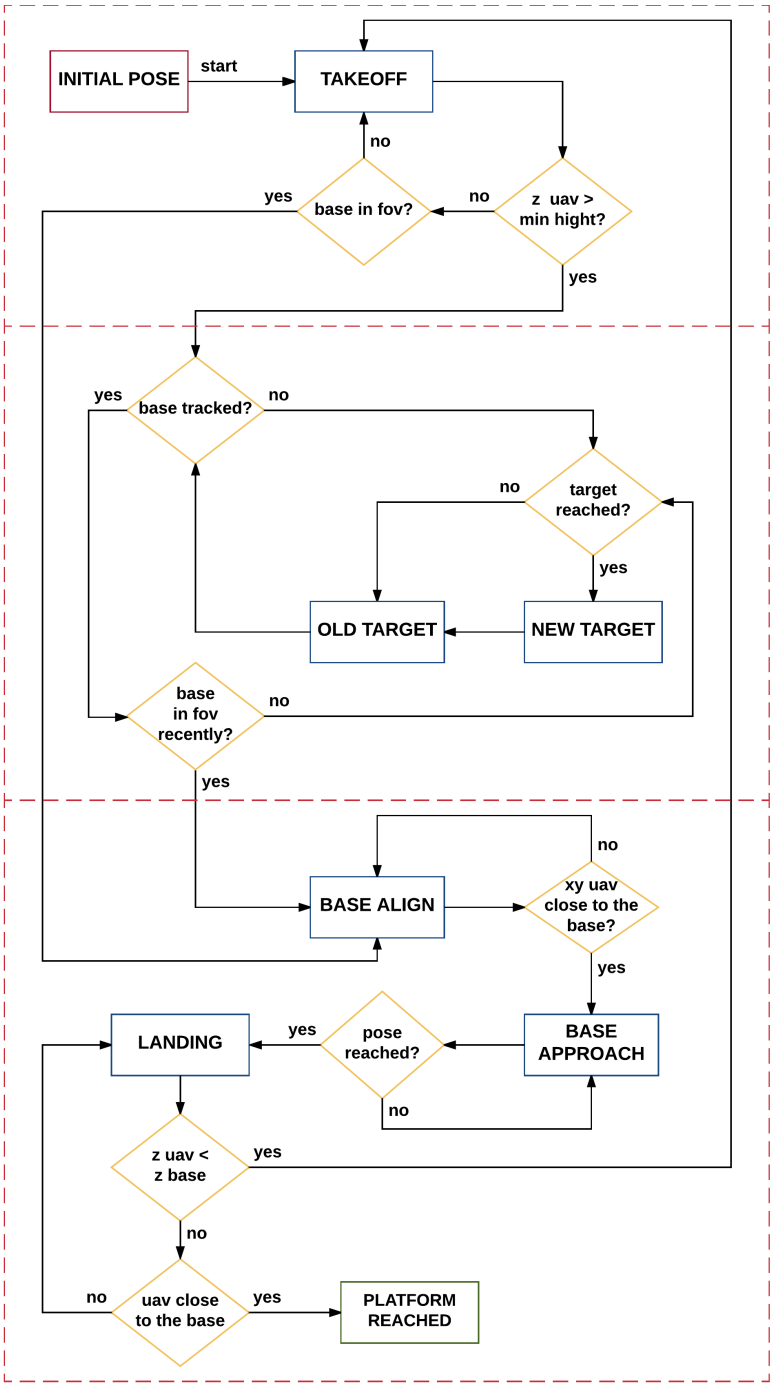


Figure 5.1: Area Exploration

Understand type of movement

From the challenge description 2.1 we know that the car is moving in a shape composed by straight lines and circumference sectors. We need to understand in which part is the platform at a given time: 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 finally 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, 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:

$$y = mx + q$$

(we perform the following analysis considering before the coordinates y_i as dependent variables, then x_i, and we are 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 (trivial solution), there will be an error between the value predicted by the line, and the observed dependent variable:

$$e_i = y_i - (mx_i + q)$$

These differences are called residuals and what we want is to find a line 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:

$$\begin{aligned} \frac{e_{tot}^2}{\partial m} &= 0 \\ \frac{e_{tot}^2}{\partial q} &= 0 \end{aligned} \tag{5.1}$$

It is easy to demonstrate that the solution of 5.1 is:

$$\begin{aligned} m &= \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \\ q &= \frac{\sum_{i=1}^{n} y_i}{n} - m\frac{\sum_{i=1}^{n} x_i}{n} \end{aligned} \tag{5.2}$$

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 following 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

$$l_{curve} = \frac{r_8 \pi}{4} \tag{5.3}$$

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

$$t_{curve} = \frac{l_{curve}}{v_{tan}} \tag{5.4}$$

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:

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

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

$$\begin{aligned} < \tilde{e_i}^2 > = \\ < (\tilde{y_i} - (mx_i + q))^2 > = \\ < \tilde{y_i}^2 - 2\tilde{y_i}(mx_i + q) + (mx_i + q)^2 > = \\ < \tilde{y_i}^2 > -2 < \tilde{y_i} > (mx_i + q) + (mx_i + q)^2 = \\ \sigma_e^2 + y_i^2 - 2y_i(mx_i + q) + (mx_i + q)^2 = \\ \sigma_e^2 + e_i^2 \end{aligned} \tag{5.5}$$

  - when we perform the linear regression on linear data the theoretical residual is

$$e_i = 0$$

And the average residual squares on the measured data

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

.

The parameter th_{line} is then:

$$th_{line} = \sqrt{\sum_{i=1}^{n} \tilde{e_i}^2} = \sqrt{\sum_{i=1}^{n} \sigma_e^2} = \sigma_e \sqrt{n} \tag{5.6}$$

– 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:

$$(\rho\cos\theta_i, \rho\sin\theta_i)$$

so the theoretical residual is:

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

To find m and q we use equation 5.2, 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 5.7

$$\begin{aligned} \lim_{n\to\infty} \frac{b-a}{n}\sum_{i=0}^{n} f(x_i) &= \int_a^b f(x)\mathrm{d}x \\ \sum_{i=0}^{n} f(x_i) &\simeq \frac{n}{b-a}\int_a^b f(x)\mathrm{d}x \end{aligned} \tag{5.7}$$

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

$$\begin{aligned} \sum_{i=1}^{n} x_i y_i &= \sum_{i=1}^{n} \rho^2\cos\theta_i\sin\theta_i \\ &\simeq \frac{n}{\theta_2-\theta_1}\rho^2\int_{\theta_1}^{\theta_2}\cos x\sin x\mathrm{d}x = \frac{n}{\theta_2-\theta_1}\frac{\rho^2}{2}\Big[-\cos^2 x\Big]_{\theta_2}^{\theta_2} \\ \sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} \rho\cos\theta_i \\ &\simeq \frac{n}{\theta_2-\theta_1}\rho\int_{\theta_1}^{\theta_2}\cos x\mathrm{d}x = \frac{n}{\theta_2-\theta_1}\rho\Big[\sin x\Big]_{\theta_2}^{\theta_2} \\ \sum_{i=1}^{n} y_i &= \sum_{i=1}^{n} \rho\sin\theta_i \\ &\simeq \frac{n}{\theta_2-\theta_1}\rho\int_{\theta_1}^{\theta_2}\sin x\mathrm{d}x = \frac{n}{\theta_2-\theta_1}\rho\Big[-\cos x\Big]_{\theta_2}^{\theta_2} \\ \sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} \rho^2\cos^2\theta_i \\ &\simeq \frac{n}{\theta_2-\theta_1}\rho^2\int_{\theta_1}^{\theta_2}\cos^2 x\mathrm{d}x = \frac{n}{\theta_2-\theta_1}\frac{\rho^2}{2}\Big[x+\cos x\sin x\Big]_{\theta_2}^{\theta_2} \end{aligned} \tag{5.8}$$

In our case we consider pieces of curve with length l_{curve} 5.3, that corresponds to a circumference arch with:

$$\rho = r_8 \qquad\qquad \theta_i \in \left[0, \frac{\pi}{4}\right] \tag{5.9}$$

We can now calculate the approximate values of m and q using 5.2 5.8 5.9:

$$m = \frac{nr_8^2\frac{n}{\pi} - r_8\frac{n2\sqrt{2}}{\pi}r_8\frac{n2(2-\sqrt{2})}{\pi}}{n\frac{nr_8^2(2+\pi)}{2\pi} - (r_8\frac{n2\sqrt{2}}{\pi})^2} = \frac{2\pi - 16\sqrt{2} + 16}{\pi^2 + 2\pi - 16}$$
$$q = \frac{r_8\frac{n2(2-\sqrt{2})}{\pi}}{n} - m\frac{r_8\frac{n2\sqrt{2}}{\pi}}{n} = r_8\frac{4 - 2\sqrt{2}(m+1)}{\pi} = r_8\bar{q} \tag{5.10}$$

Now we can calculate the theoretical average residual square, using again the approximations 5.8:

$$\begin{aligned} < e_i^2 > &= \frac{\sum_{i=1}^{n}\Big(r_8\sin\theta_i - (mr_8\cos\theta_i + q)\Big)^2}{n} \\ &= \frac{\sum_{i=1}^{n}\Big(r_8^2\sin^2\theta_i - 2r_8\sin\theta_i(mr_8\cos\theta_i + q) + (mr_8\cos\theta_i + q)^2\Big)}{n} \\ &= \frac{\sum_{i=1}^{n} r_8^2\xi}{n} = r_8^2\xi \\ \xi &= \frac{\pi - 2 + m^2(\pi+2) + 2\pi\bar{q}^2 - 4m - 8\bar{q}(2-\sqrt{2}) + 8m\bar{q}\sqrt{2}}{2\pi} \end{aligned}$$

Finally we calculate

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

.
The parameter th_{curve} is:

$$th_{curve} = \sqrt{\sum_{i=1}^{n}\tilde{e_i}^2} = \sqrt{\sum_{i=1}^{n}\rho^2\xi + \sigma_e^2} = \sqrt{n}(\sigma_e + \rho\sqrt{\xi}) \tag{5.11}$$

The figure 5.2 shows the typical evolution of the total residual during this first phase: the different phases of linear and circular movement can be detect in the graph. Furthermore the point of regime change can be seen both in figure 5.2 and in the map 5.3 in which also all the estimate positions of the base are plotted.

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 phases 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 $v_{tan}t_s$.

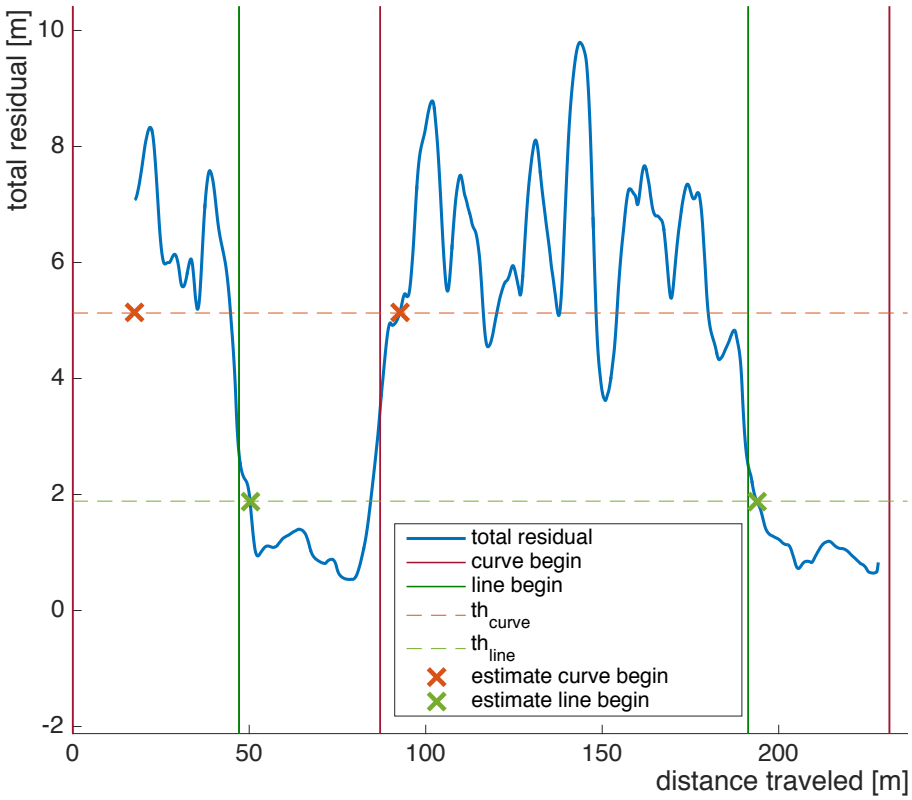


Figure 5.2: Evolution of the total residual during this first phase (in blue). The vertical lines are the real moments in which the car changes movement types: green a linear phase starts, red a circular phase begins. The horizontal lines are the thresholds for the detection of the two different regimes. The cross are the moment in which the algorithm understands the change.

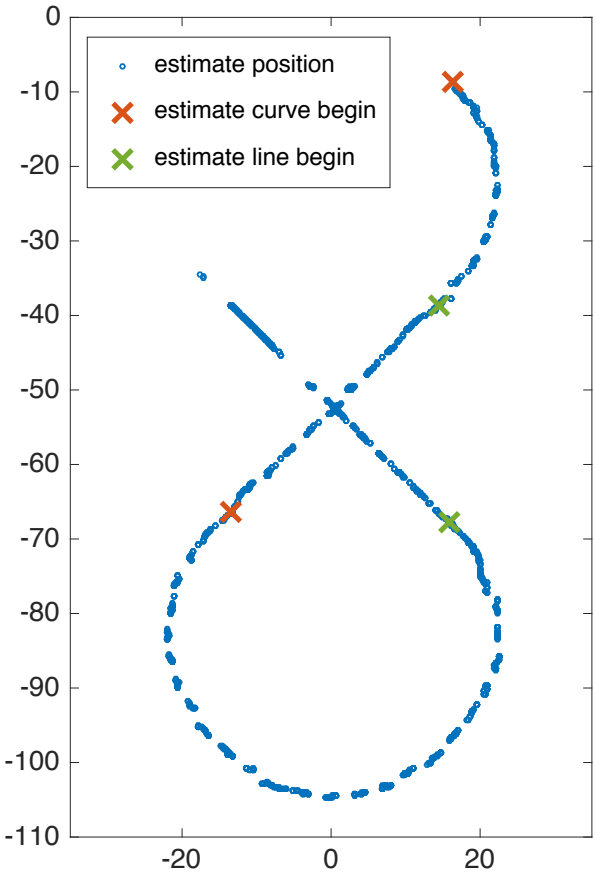


Figure 5.3: Map of the estimated position of the car in blue. The cross are the moments in which the algorithm understands the change. Red crosses from line to curve. Green crosses from curve to line.

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

$$\begin{cases} x_1 &= x_0 + v_{tan} t_s \cos\theta_0 \\ y_1 &= y_0 + v_{tan} t_s \sin\theta_0 \\ \theta_1 &= \theta_0 \end{cases} \tag{5.12}$$

- 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

$$l_{curve} = \rho|\beta_s|$$

  To find the angle β_s that the platform will span in t_s seconds. In our case:

$$\begin{aligned} v_{tan} t_s &= r_8|\beta_s| \\ |\beta_s| &= \frac{v_{tan} t_s}{r_8} \end{aligned} \tag{5.13}$$

  The final angle will be:

$$\theta_1 = \theta_0 + \beta_s \tag{5.14}$$

  The segment connecting (x_0, y_0) and (x_1, y_1) has 5.4:

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

$$\theta_{chord} = \theta_0 + \frac{\theta_0 + \theta_1}{2} \tag{5.15}$$

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

$$l_{chord} = 2r_8 \sin\frac{|\beta_s|}{2} \tag{5.16}$$

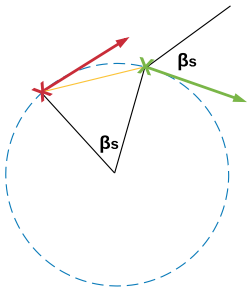


Figure 5.4: In red position of initial state at time t_0. In green final estimate state at time t_1. In yellow the chord between the two states with length l_{chord} and orientation θ_{chord}.

In order to properly find the final point (x_1, y_1) we have to resolve an other last problem: both β_s and $-\beta_s$ span a curve of length $v_{tan}t$, 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 5.13 5.15 5.16:

$$\begin{cases} x_1^a &= x_0 + l_{chord} \cos\left(\theta_0 + \frac{|\beta_s|}{2}\right) \\ y_1^a &= y_0 + l_{chord} \sin\left(\theta_0 + \frac{|\beta_s|}{2}\right) \\ \theta_1^a &= \theta_0 + |\beta_s| \end{cases} \tag{5.17}$$

$$\begin{cases} x_1^b &= x_0 + l_{chord} \cos\left(\theta_0 - \frac{|\beta_s|}{2}\right) \\ y_1^b &= y_0 + l_{chord} \sin\left(\theta_0 - \frac{|\beta_s|}{2}\right) \\ \theta_1^b &= \theta_0 - |\beta_s| \end{cases} \tag{5.18}$$

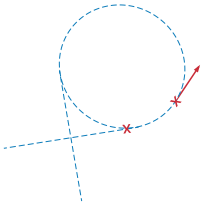
In order to understand which is the correct state we can calculate the distance between the two possible final points and a point of the trajectory estimated at time $t_{-\alpha} < t_0$. For sure the state with smaller distance will be the right final state, because the wrong one leads to a position further away.
The images 5.5 show all the passages we perform to find the right final state.

The image 5.6 shows where the algorithm calculates the way points for the quadrotor in order to following the moving car.

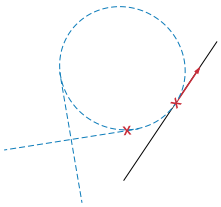
Following the base and proceed with following phase

At this point we can use the predict position of the platform to control the quadrotor in order to following the base and at the right moment proceed with the other phases.
The right moment to perform the landing is at the start of a line segment:

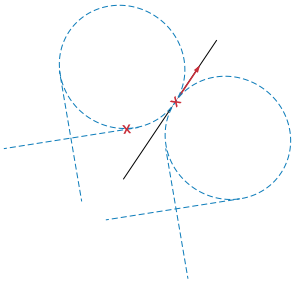
- if we first detect the base and we understand that it is moving in a circumference we cannot land, we can follow the base and waiting when we will detect a change in the regime from curve to line. At this point we can proceed with the landing.
- if we first detect the base and we understand that it 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 from the line movement to the curve. At this point we can calculate where the next change point will be:
  - we know the orientation of the straight line portion just finished and it is θ_{line}



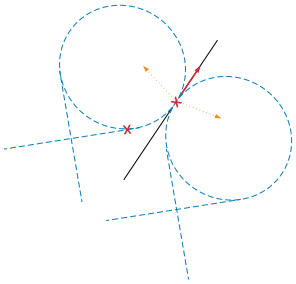
(a) Red cross with arrow: state a t_0. Red arrow: current velocity vector. Red cross: previous position a $t_{-\alpha}$.



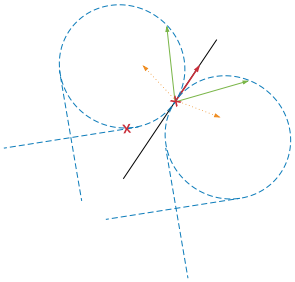
(b) Black line: direction of the velocity vector, with angle θ_0. We have symmetry with respect to this axes.



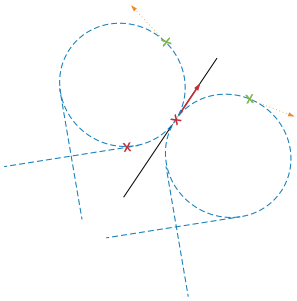
(c) Blue lines: real and symmetric path. We do not know which of the two trajectories is correct.



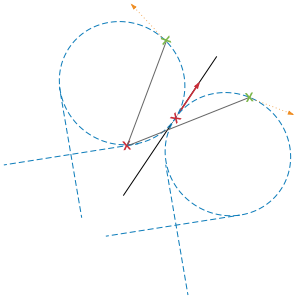
(d) Yellow arrows: estimated future angles θ_1^a and θ_1^b that the base will assume at time t_1.



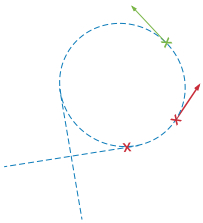
(e) Green arrows: segment from state at t_0 and possible states at t_1: with angles θ^a_{chord} and θ^b_{chord} and length l_{chord}



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

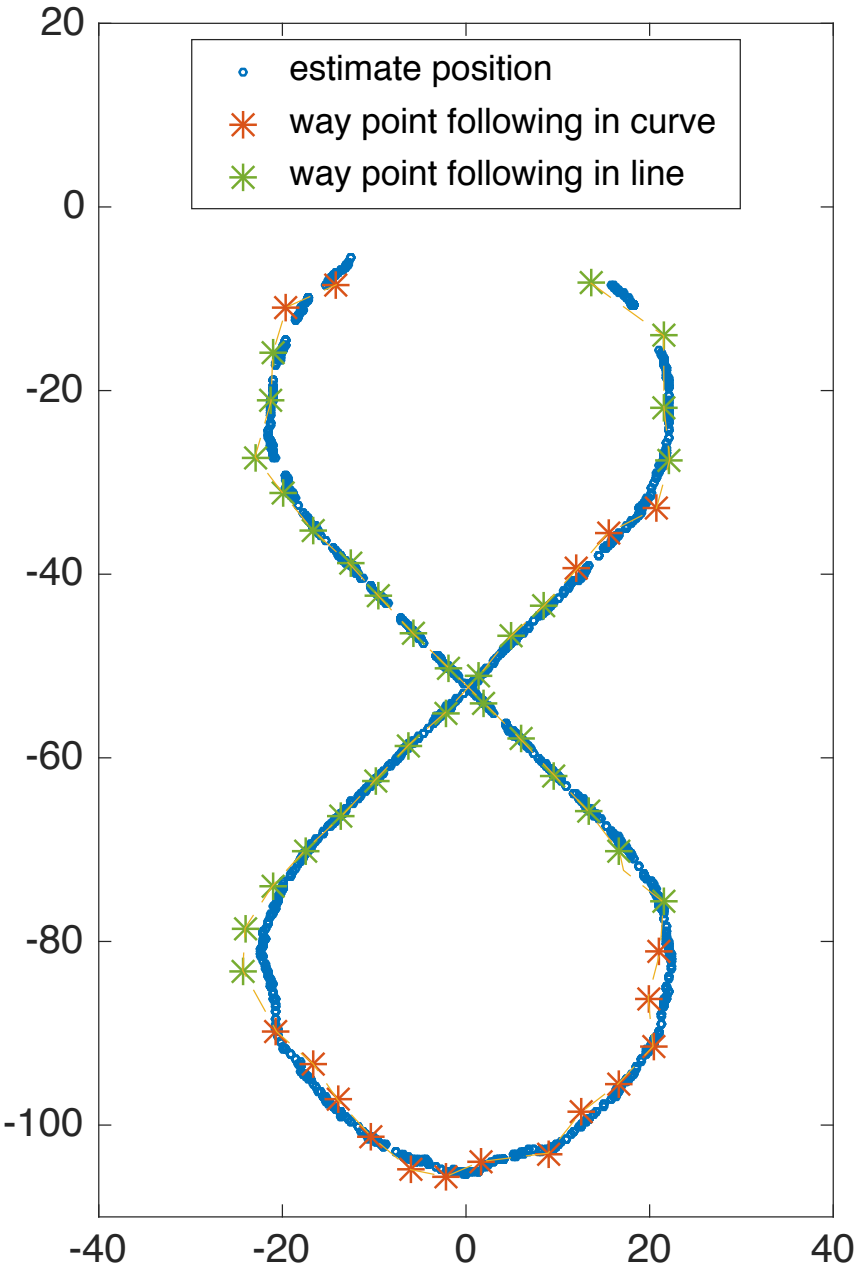


Figure 5.6: Data from the first phase of the area exploration. Blue circles position of the base estimate. Stars the position in which the quadrotor should go to following the base until a proper moment to proceed with the following phases is detected.

– 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 5.15 5.16 we know that the segment connecting the change point and the future intersection point has length $\sqrt{2}r_8$ and angle $\theta_{line} \pm \frac{3\pi}{4}$

– we can apply the same method described before to find the two possible intersection points:

$$\begin{cases} x^a_{intersection} &= x_{changing} + \sqrt{2}r_8 \cos\left(\theta_{line} + \frac{3\pi}{4}\right) \\ y^a_{intersection} &= y_{changing} + \sqrt{2}r_8 \sin\left(\theta_{line} + \frac{3\pi}{4}\right) \\ \theta^a_{intersection} &= \theta_{line} + \frac{3\pi}{2} \end{cases} \tag{5.19}$$

$$\begin{cases} x^b_{intersection} &= x_{changing} + \sqrt{2}r_8 \cos\left(\theta_{line} - \frac{3\pi}{4}\right) \\ y^b_{intersection} &= y_{changing} + \sqrt{2}r_8 \sin\left(\theta_{line} - \frac{3\pi}{4}\right) \\ \theta^b_{intersection} &= \theta_{line} - \frac{3\pi}{2} \end{cases} \tag{5.20}$$

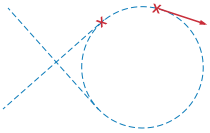
and select the right one with minimum distance with the current estimate position of the platform. The images 5.7 show all the passages we perform to find the right intersection point.

Now that we know where the next change of regime will be, we can follow the base and continue with the following phases when the base is close to the intersection point.

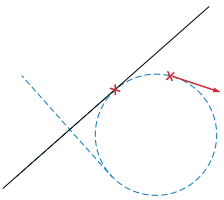
5.1.2 Second phase - Approaching the base

5.1.3 Second phase - Following the base

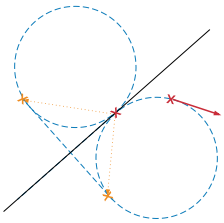
5.1.4 Second phase - Landing on the base



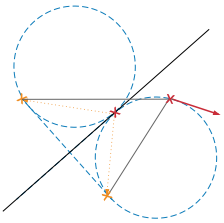
(a) Red cross with arrow: state a t_0. Red arrow: current velocity vector. Red cross: changing point from line to curve.



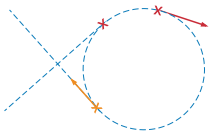
(b) Black line: direction of the line sector just finished. The direction is taken as the slope of the best linear fit found in the previous regime.



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

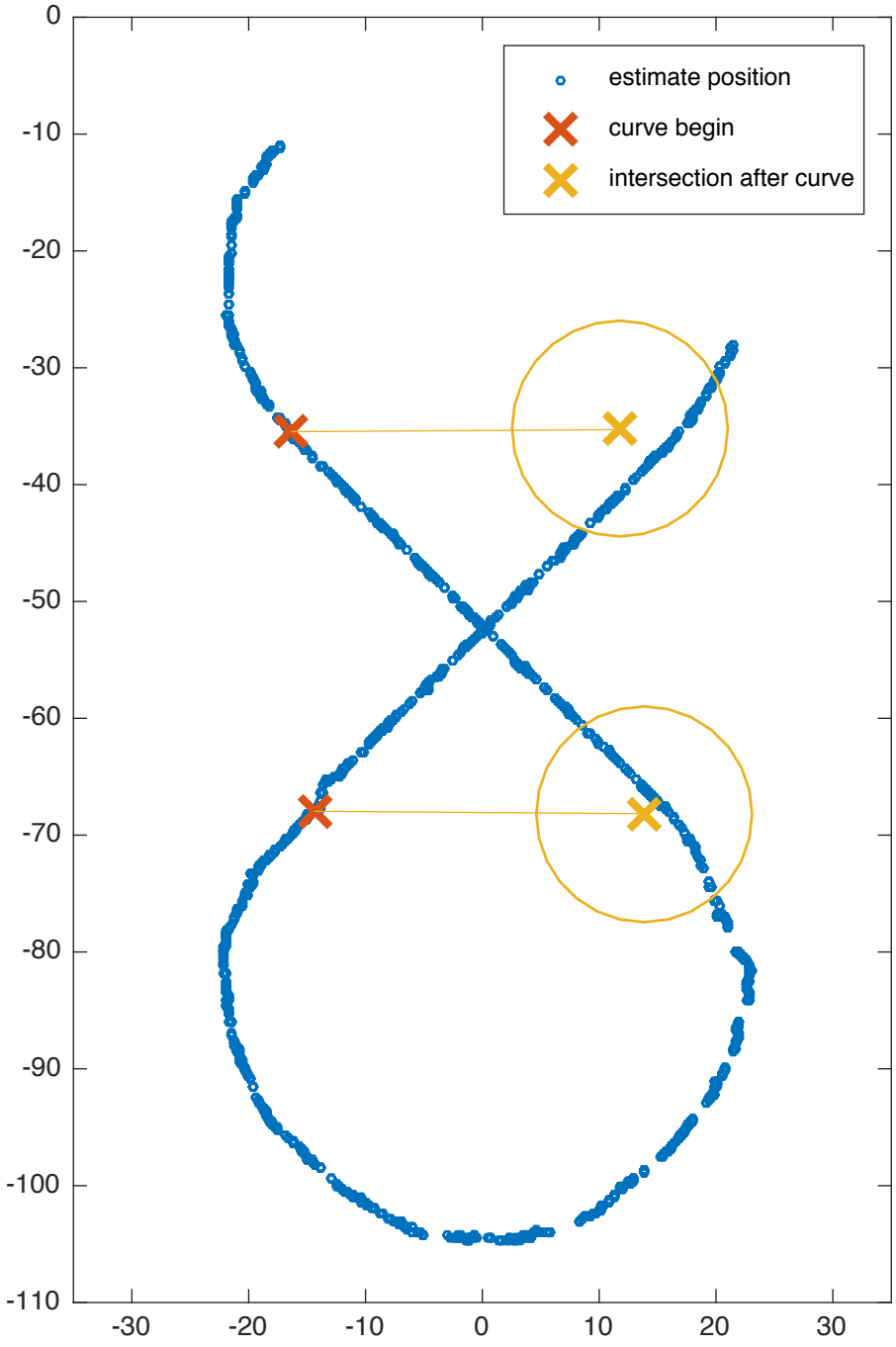


Figure 5.8: Data from the first phase of the area exploration. Blue circles position of the base estimate. Red cross the points in which the algorithm detect a passage from a line phase to a curve. Green cross from a curve to a line. Yellow cross the position where the quadrotor should go in order to intersect the platform when is about to start a line phase.

Chapter 6

Trajectory Generator

6.1 Base trajectory prediction

6.2 Rapid Trajectory

how to compute the acceleration... comparision between diff imu and thrust

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 compute the acceleration.
There are several ways to make this calculation:

- IMU measurements:
- finite difference of velocity:
- total thrust:

6.3 Results

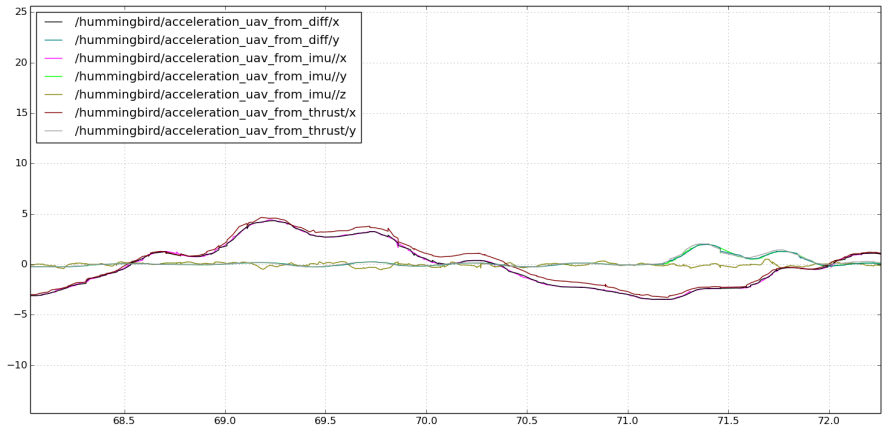


Figure 6.1: Comparison Acceleration

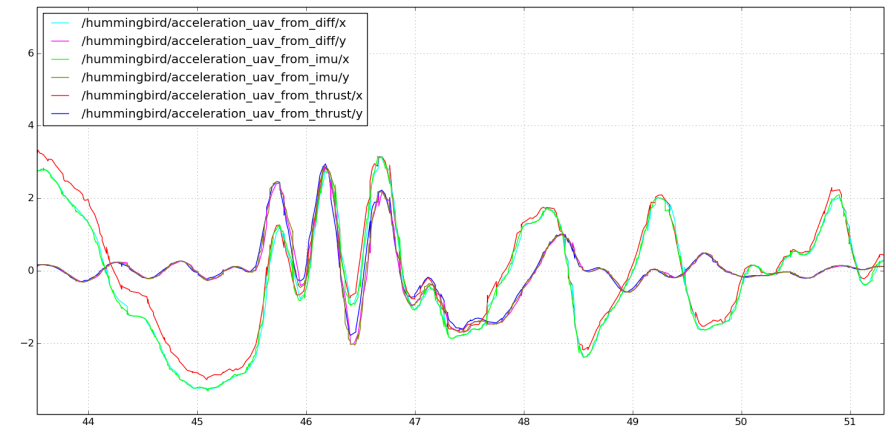


Figure 6.2: Comparison Acceleration Different Mass

Chapter 7

Experiments

Provide numerical results, plots, and timings. Interpret the data.

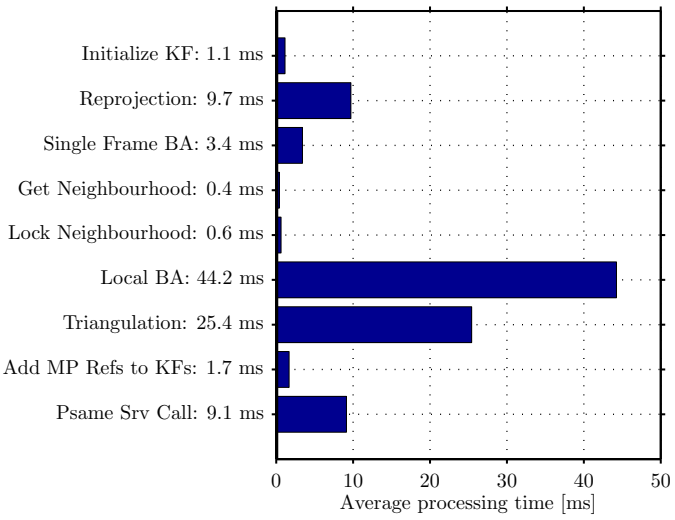


Figure 7.1: Example of a figure.

Chapter 8

Discussion

Explain both the advantages and limitations of your approach.

8.1 Conclusion

Summarize your work and what came out of it.

8.2 Future Work

How would you extend the work? Can you propose another approach?

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] MBZIRC. Mbzirc challenge description, 2016.

[14] 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.

[15] Wikipedia. Runge-kutta methods, 2016.

[16] Steven S. Beauchemin and John L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995.

[17] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 81(1):674–679, 1981.

[18] 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.

[19] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *IEEE Transactions on pattern analysis and machine intelligence*, 21(8):774–780, 1999.

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

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

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

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

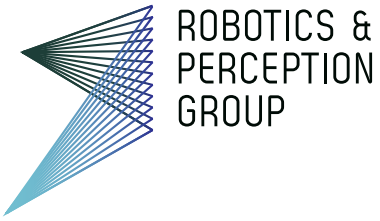
[24] Salinas Bence Magyar Hamdi Sahloul, Rafael Munoz. Ar-sys ros package, 2014.

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

[26] 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.

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

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



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, 31. 8. 2016: _____