Vision-based Autonomous Landing of a Quadcopter with Field-of-View Constraints

by

Sequoyah Walters

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science

(Mechanical Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2023

Date of final oral examination: 08/14/2023

Committee members:

Xiangru Xu, Assistant Professor, Mechanical Engineering
Peter Adamczyk, Associate Professor, Mechanical Engineering
Dan Negrut, Professor, Mechanical Engineering

ACKNOWLEDGMENTS

I would like to thank Professor Xiangru Xu, my advisor, for his invaluable guidance, support and insightful feedback during my time at UW. I express my appreciation to the members of ARC Lab for their friendship and mentorship. To my family, friends, and especially my girlfriend, Sami, thank you for your unwavering support, belief in my abilities, and understanding. Your presence in my life has been a constant source of motivation, and I am forever grateful.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This thesis addresses the problem of autonomous and precise landing of a quadcopter with field-of-view (FOV) constraints. For quadcopter precision-landing tasks, knowledge of the landing pad location is vital. When landing pad location is obtained by a quadcopter-mounted camera, landing accuracy can be improved if the quadcopter maintains visual contact with the landing pad at all times. The main contribution of this work is the integration of FOV constraints into the minimum-snap trajectory generation algorithm for vision-based quadcopter landing. Verification in Gazebo simulation and with real-world experiments is done to show the efficacy of the algorithm, in which visual-inertial odometry (VIO) is used for estimating the position and orientation of the quadcopter. A custom-designed quadcopter is used to perform real-world tests, in which all navigation and control algorithms are computed onboard in real time. An open-source implementation of the code is provided to the community, which also includes instructions for building the VIO quadcopter and experimentation.

1 INTRODUCTION

1.1 Motivation

In the rapidly evolving domain of unmanned aerial vehicles (UAVs), quadcopters have emerged as versatile tools with many applications. The success of many of these applications hinges on the ability of quadcopters to execute precise and safe landings, motivating the need for development of robust landing algorithms. Real-world examples include situations where quadcopters must safely touch down in close proximity to individuals, providing critical roles in tasks such as delivery services and search-and-rescue operations. Similarly, collaborative landing scenarios require quadcopters to land on moving robotic platforms. Within this realm, the focus of this thesis lies in FOV constrained quadcopter landing, in which landing accuracy is improved by ensuring visibility of the landing pad by a quadcopter-mounted camera.

The challenges posed by FOV constrained landings are multifaceted. Factors such as quadcopter attitude[1] and position relative to the landing pad significantly influence line of sight of the camera. Inadequate visibility may result in increased landing errors, collisions, and a decrease in overall safety. Additionally, trajectory planning and control must be run in real-time on the low-power onboard computer, requiring light-weight and efficient algorithms. Addressing these challenges is imperative to ensure the reliability

[1] Attitude represents the orientation of an aerial vehicle.

and safety of quadcopter operations within real-world contexts.

1.2 Overview

This work aims to use a quadcopter equipped with a forward-facing stereo camera and a down-facing monocular camera for autonomous landing. The stereo camera paired with an IMU is used to perform VIO to estimate the quadcopter state (pose[2] and velocity), while the down-facing camera is used to track an AprilTag fiducial marker on a landing pad.

The AprilTag detection algorithm tracks the pose of the AprilTag marker in the frame of the down-facing camera. Once the AprilTag marker is detected, a quadratic program (QP) is solved to find a minimum-snap[3] trajectory from the current quadcopter pose to the pose of the landing pad (i.e. AprilTag marker). The QP contains constraints that enforce the center of the AprilTag marker to be in the FOV of the down-facing camera. Formulating these FOV constraints in the QP framework constitutes the main theoretical contribution of this work.

After a minimum-snap trajectory has been obtained, we use a model-predictive control (MPC) algorithm to track (i.e. follow) the trajectory. MPC provides thrust and angular rate to control the quadcopter, and uses a 10-dimensional quadcopter model in which the state is given by position $\boldsymbol{r}$, velocity $\boldsymbol{v}$ and orientation quaternion $\boldsymbol{q}$. Perception-aware MPC (PAMPC)

[2]Pose represents the 3D position *and* orientation of an object.

[3]Snap is the second time derivative of acceleration.

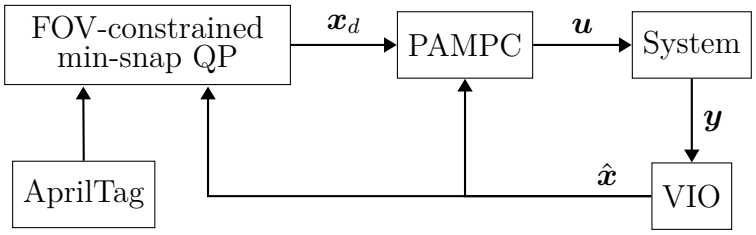

Figure 1.1: Implemented feedback loop for landing on an AprilTag marker. The AprilTag algorithm estimates the pose of the AprilTag marker. Visual-inertial odometry (VIO) provides a quadcopter state estimate $\hat{\boldsymbol{x}}$ given the system output $\boldsymbol{y}$ (stereo image and IMU data). The FOV-constrained min-snap QP provides the desired state $\boldsymbol{x}_d$ given $\hat{\boldsymbol{x}}$ and AprilTag marker pose. Perception-aware model-predictive control (PAMPC) generates the control input $\boldsymbol{u}$ given $\boldsymbol{x}_d$ and $\hat{\boldsymbol{x}}$.

from [1] is implemented by including a perception term in the MPC objective function to ensure the down-facing camera can view the landing pad.

A companion computer (CC) mounted on the quadcopter receives data streams from all the components on the quadcopter such as the cameras, IMU and flight control unit (FCU). The CC is the "brain" of the quadcopter and is responsible for computing 1) the VIO-provided state estimation, 2) the landing pad pose and 3) control inputs via PAMPC. An illustration of the proposed framework is shown in Figure 1.1. Code for generating the FOV-constrained landing trajectory and for MPC/PAMPC control of the quadcopter has been released as open-source software[4].

[4] `https://github.com/seqwalt/vioquad_land`

2 BACKGROUND

This chapter provides the necessary context to understand the proposed research. Quadcopter dynamics and control are first discussed, followed by an introduction to the minimum-snap QP trajectory generation framework. Then, VIO methods and variations commonly used with quadcopter system are described. Finally, the AprilTag detection algorithm is introduced, along with reasoning for its use in this thesis.

2.1 Quadcopter Dynamics

We consider the following 10-dimensional quadcopter model [1] in this work. The state of the quadcopter is

$$\begin{aligned}\boldsymbol{x} &= \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & q_w & q_x & q_y & q_z \end{bmatrix}^T \in \mathbb{R}^{10} \\ &= \begin{bmatrix} \boldsymbol{r}^T & \boldsymbol{v}^T & \boldsymbol{q}^T \end{bmatrix}^T,\end{aligned}$$

where we define $\boldsymbol{r} := [x \;\; y \;\; z]^T$, $\boldsymbol{v} := [\dot{x} \;\; \dot{y} \;\; \dot{z}]^T$ and $\boldsymbol{q} := [q_w \;\; q_x \;\; q_y \;\; q_z]^T$. The vectors $\boldsymbol{r}$ and $\boldsymbol{v}$ are the position and velocity of the quadcopter body frame expressed in the world frame, respectively. The orientation vector $\boldsymbol{q}$ is a unit quaternion that defines the rotation from the world frame to the body frame. The control inputs are mass-normalized thrust τ and angular

velocity $\boldsymbol{\omega}$ expressed in the body frame, compactly expressed as

$$\boldsymbol{u} = \begin{bmatrix} \tau & \boldsymbol{\omega}^T \end{bmatrix}^T \in \mathbb{R}^4.$$

The system dynamics can then be described as follows:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ &= \begin{bmatrix} \dot{\boldsymbol{r}} \\ \dot{\boldsymbol{v}} \\ \dot{\boldsymbol{q}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{g} + Rot(\boldsymbol{q})\boldsymbol{\tau} \\ \frac{1}{2}\Lambda(\boldsymbol{\omega})\boldsymbol{q} \end{bmatrix}, \end{aligned} \tag{2.1}$$

where $\boldsymbol{g} := [0 \quad 0 \quad -g]$ is gravity expressed in the world frame and $\boldsymbol{\tau} = [0 \quad 0 \quad \tau]^T$ is mass-normalized thrust expressed in the body frame. The operator $Rot(\cdot)$ converts quaternions to rotation matrices according to

$$Rot(\boldsymbol{q}) := \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2\left(q_x q_y + q_w q_z\right) & 2\left(q_x q_z - q_w q_y\right) \\ 2\left(q_x q_y - q_w q_z\right) & 1 - 2q_x^2 - 2q_z^2 & 2\left(q_y q_z + q_w q_x\right) \\ 2\left(q_x q_z + q_w q_y\right) & 2\left(q_y q_z - q_w q_x\right) & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix}, \tag{2.2}$$

and the operator $\Lambda(\cdot)$ is defined as

$$\Lambda(\boldsymbol{\omega}) := \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}.$$

2.2 Quadcopter Control

Quadcopters are mechanically simple machines that can achieve fast and agile motion. However, they are underactuated control systems due to having

6 degrees of freedom (i.e. position and orientation) with only four control inputs (i.e. four motors), making the control of quadcopters challenging. Here we describe MPC, a versatile optimization-based method that can be used for quadcopter control.

Model Predictive Control

MPC is a method that can be used for quadcoptor control to track a desired reference trajectory $\boldsymbol{x}_d(t)$, given current state estimate $\hat{\boldsymbol{x}}(t_0)$ at time t_0. To use MPC for continuous nonlinear systems like (2.1), discretization is required.

Let $F(\boldsymbol{x}, \boldsymbol{u})$ be a discretization of (2.1) (using the multiple-shooting technique) with time step $\Delta t := t_i - t_{i-1},\ \forall i \in \{1, \dots, N\}$, and define $\boldsymbol{z}_N := \Big[\boldsymbol{x}(t_N) - \boldsymbol{x}_d(t_N)\Big]$ and $\boldsymbol{z}(t) := \begin{bmatrix} \boldsymbol{x}(t) - \boldsymbol{x}_d(t) \\ \boldsymbol{u}(t) \end{bmatrix}$. The MPC method then requires iteratively solving a finite-horizon optimization problem:

$$
\begin{aligned}
\min_{\substack{\boldsymbol{x}(t_1), \dots, \boldsymbol{x}(t_N) \\ \boldsymbol{u}(t_0), \dots, \boldsymbol{u}(t_{N-1})}} \quad & \boldsymbol{z}_N^T Q_N \boldsymbol{z}_N + \sum_{i=0}^{N-1} \boldsymbol{z}(t_i)^T Q \boldsymbol{z}(t_i) \qquad (2.3) \\
\text{s.t.} \quad & \boldsymbol{x}(t_0) = \hat{\boldsymbol{x}}(t_0) \\
& \boldsymbol{x}(t_{i+1}) = F(\boldsymbol{x}(t_i), \boldsymbol{u}(t_i)), \quad \forall i \in \{0, \dots, N-1\} \\
& \boldsymbol{x}(t_i) \in \mathcal{X}, \quad \forall i \in \{0, \dots, N\} \\
& \boldsymbol{u}(t_i) \in \mathcal{U}, \quad \forall i \in \{0, \dots, N-1\}
\end{aligned}
$$

where Q and Q_N denote weight matrices and $\mathcal{X} \in \mathbb{R}^{10}$ and $\mathcal{U} \in \mathbb{R}^4$ are

constraints on the state and control input respectively. The optimization problem is re-solved at a high rate to ensure accurate trajectory tracking. Each time the problem is solved, $\boldsymbol{u}(t_0)$ is applied as a control input to the system. Specifically, control inputs given by MPC are sent to the FCU which runs PX4 flight software [2]. The PX4 attitude rate PID controller then computes motor voltages to directly control the motors.

2.3 Minimum-Snap Trajectory Generation

At its core, the minimum-snap trajectory generation method proposed in [3] consists of solving a QP in which the decision variables control the shape of a piecewise polynomial trajectory, the objective function minimizes the snap (i.e. fourth derivative of position) of the trajectory, and the constraints enforce continuity and satisfaction of the boundary conditions and waypoints of the trajectory.

From [3], $\boldsymbol{\sigma_T(t)}$ is the quadcopter trajectory and is defined as

$$\boldsymbol{\sigma}_T(t) = \begin{bmatrix} \mathbf{r}_T(t) \\ \psi_T(t) \end{bmatrix} = \begin{bmatrix} x_T(t) \\ y_T(t) \\ z_T(t) \\ \psi_T(t) \end{bmatrix} := \begin{cases} \sum_{i=0}^{n} \boldsymbol{\sigma}_{Ti1} t^i & t_0 \leq t < t_1 \\ \sum_{i=0}^{n} \boldsymbol{\sigma}_{Ti2} t^i & t_1 \leq t < t_2 \\ \quad \vdots & \\ \sum_{i=0}^{n} \boldsymbol{\sigma}_{Tim} t^i & t_{m-1} \leq t \leq t_m \end{cases}, \quad (2.4)$$

where $\psi_T(t)$ is the yaw trajectory, n is the polynomial order, and m is the number of time intervals.

To enforce continuity of the p-th derivative of $\boldsymbol{\sigma}_T$, where $\boldsymbol{\sigma}_T^{(p)}(t) =$

$\sum_{i=p}^{n} \frac{i!}{(i-p)!}\boldsymbol{\sigma}_{T_{ij}} t^{i-p}$, $\forall t \in [t_{j-1},\ t_j]\,,\ j \in \{1,\ldots,m-1\}$, we can write:

$$\sum_{i=p}^{n} \frac{i!}{(i-p)!}\boldsymbol{\sigma}_{T_{ij}} t_j^{i-p} = \sum_{i=p}^{n} \frac{i!}{(i-p)!}\boldsymbol{\sigma}_{T_{i,j+1}} t_j^{i-p}$$
$$\sum_{i=p}^{n} \boldsymbol{\sigma}_{T_{ij}} - \boldsymbol{\sigma}_{T_{i,j+1}} = 0, \quad \forall j \in \{1,\ldots,m-1\}. \tag{2.5}$$

Defining the notation $\mathbf{r}_T^{(p)}(t_j) := \left.\frac{d^p \mathbf{r}_T}{dt^p}\right|_{t=t_j}$ and $\psi_T^{(p)}(t_j) := \left.\frac{d^p \psi_T}{dt^p}\right|_{t=t_j}$, the minimum-snap QP can then be formulated as

$$\min_{\substack{\boldsymbol{r}_{T_{ij}},\ \psi_{T_{ij}} \\ \forall i \in \{0,\ldots,n\} \\ \forall j \in \{1,\ldots,m\}}} \quad \int_{t_0}^{t_m} \mu_r \left\|\mathbf{r}_T^{(k_r)}(t)\right\|^2 + \mu_\psi \left(\psi_T^{(k_\psi)}(t)\right)^2 dt \tag{2.6}$$

$$\text{s.t.} \quad \boldsymbol{\sigma}_T(t_j) = \boldsymbol{\sigma}_j, \quad j = 0,\ldots,m \tag{2.7}$$
$$\mathbf{r}_T^{(p)}(t_j) = \mathbf{r}_j^{(p)} \text{ or free}, \quad j = \{0,m\};\ p = 1,\ldots,k_r \tag{2.8}$$
$$\psi_T^{(p)}(t_j) = \psi_j^{(p)} \text{ or free}, \quad j = \{0,m\};\ p = 1,\ldots,k_\psi \tag{2.9}$$
$$\sum_{i=p}^{n} (\boldsymbol{r}_{T_{ij}} - \boldsymbol{r}_{T_{i,j+1}}) = 0, \quad j = 1,\ldots,m-1;\ p = 0,\ldots,k_r \tag{2.10}$$
$$\sum_{i=p}^{n} (\psi_{T_{ij}} - \psi_{T_{i,j+1}}) = 0, \quad j = 1,\ldots,m-1;\ p = 0,\ldots,k_\psi \tag{2.11}$$

where μ_r and μ_ψ make the integrand nondimensional. Equality (2.7) encodes way-point constraints, (2.8) and (2.9) encode constraints at the initial and final time (e.g. to fix initial velocity and acceleration), (2.10) and (2.11) ensure continuity of the piecewise polynomials and their derivatives (derived from (2.5)). In [3], $k_r = 4$ to minimize trajectory snap and $k_\psi = 2$ to minimize yaw angular acceleration.

Note (2.6)–(2.11) can be written in the following compact form of a convex QP:

$$\begin{aligned} \min_{\boldsymbol{c}} \quad & \boldsymbol{c}^T H \boldsymbol{c} + \boldsymbol{c}^T \boldsymbol{g} \\ \text{s.t.} \quad & A\boldsymbol{c} \leq \boldsymbol{b}, \end{aligned} \tag{2.12}$$

where $\boldsymbol{c}$, $\boldsymbol{g}$, $\boldsymbol{b}$, A and $H \in \mathbb{S}_+$ are of appropriate dimension and $\mathbb{S}_+$ denotes the set of symmetric positive semidefinite matrices.

Analytical expression of the objective function

Noting that (2.6) has an integral objective function, we show here how to obtain an analytical form. Since we are minimizing a quadratic objective, we first express the square of the trajectory for the p-th derivative $\boldsymbol{\sigma}_T^{(p)}(t)$. For example $p = 4$ would be used for snap minimization. To keep generality, we do not specify the value of p in the following.

$$\begin{aligned} \left(\boldsymbol{\sigma}_T^{(p)}(t)\right)^2 &= \left(\sum_{i=p}^{n} \frac{i!}{(i-p)!} \boldsymbol{\sigma}_{T_{ij}} t^{i-p}\right)^2 \\ &= \sum_{i=p}^{n}\sum_{k=p}^{n} \frac{i!k!}{(i-p)!(k-p)!} \boldsymbol{\sigma}_{T_{ij}} \boldsymbol{\sigma}_{T_{kj}} t^{i+k-2p} \quad \forall t \in [t_{j-1},\ t_j],\ j \in \{1, \ldots, m\} \end{aligned}$$

We can then take the integral to evaluate the objective:

$$\int_{t_{j-1}}^{t_j} \left(\boldsymbol{\sigma}_T^{(p)}(t)\right)^2 dt = \sum_{i=p}^{n}\sum_{k=p}^{n} \frac{i!\,k!\,\boldsymbol{\sigma}_{T_{ij}}\boldsymbol{\sigma}_{T_{kj}}}{(i-p)!(k-p)!} \frac{t_j^{i+k-2p+1} - t_{j-1}^{i+k-2p+1}}{i+k-2p+1} \quad \forall j \in \{1,\ldots,m\}$$

Recall that n is the order of the piecewise polynomials, m is the number of time intervals and p is the order for minimization ($p = 4$ for minimum snap). By putting the objective function into this summation form, we can easily load the H matrix from (2.12) with the required terms.

2.4 Fiducial Marker: AprilTag

The AprilTag 3 fiducial system [4] is a popular choice for UAV landing applications due to its ability to trade-off between pose estimation accuracy and estimation speed via simple parameter tuning. This capability is especially useful for low size weight and power applications such as UAVs that only have onboard computing capabilities such as is considered in this work. Additionally, the AprilTag algorithm reduces random noise interference via adaptive thresholding, and improves pose estimation accuracy by using edge refinement methods.

As the altitude of the landing quadcopter decreases, the edges of the AprilTag marker will exceed the FOV of the down-facing camera. As such, it is important to include multiple markers of differing sizes on the landing pad.

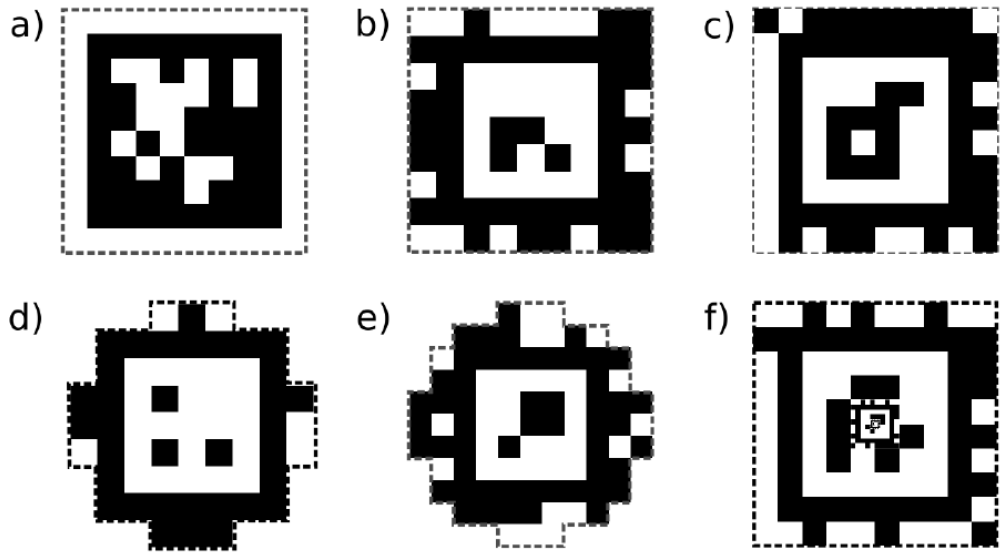

Figure 2.1: Different types of fiducial markers that can be used by the AprilTag 3 algorithm [4]. In this work, the recursive marker of type f) is used due to its applicability for quadcopter landing.

A recursive pattern of concentric markers (see Figure 2.1) as introduced in [4] is convenient for ensuring a small marker footprint, and enabling precise quadcopter landing in the center of the marker pattern. The general outline of the AprilTag algorithm is as follows:

1. Apply adaptive thresholding to the grayscale input image.
2. Segment continuous boundaries.
3. Fit quadrilaterals to each cluster of unordered boundary points.
4. Decoding the tag.
5. Perform edge refinement.

2.5 Visual-Inertial Odometry

In scenarios in which global positioning information (using GPS, motion-capture) is not available, autonomous quadcopters must rely on other methods for state estimation. Given sensor data such as pixel intensities from a camera and IMU measurements, visual-inertial odometry (VIO) can be used for quadcopter state estimation. In this section, perspective camera modelling is discussed, visual odometry (VO) and VIO are introduced, then a short review of relevant VIO research is provided.

Modelling a Perspective Camera

In VO, the pinhole camera representation is often used to provide a simple approximation of a perspective camera. A true pinhole camera does not contain a lens, but instead a small aperture (i.e. the pinhole) for light to pass through to a sensor (see Figure 2.2).

In order for visual odometry algorithms to leverage the simplicity of the pinhole model, the raw camera image first needs to be undistorted such that straight lines in a 3D scene remain straight in the image. Common distortion types are shown in Figure 2.3.

Let $P = [X,\ Y,\ Z]^T$ be a 3D point in the camera reference frame C, as shown in Figure 2.2. P is projected onto the 2D image plane with

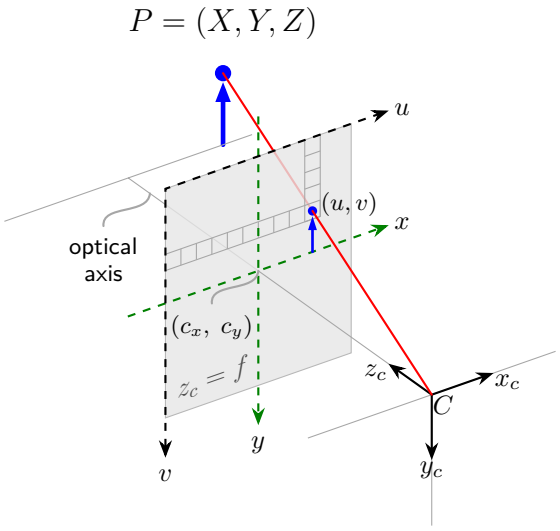

Figure 2.2: Pinhole camera model. The pinhole is located at the origin of camera frame C. Note the image plane is located a distance f (focal length) in front of the pinhole. The principal point is located along the optical axis at pixel coordinates (c_x, c_y). The 3D point P is expressed in the camera frame.

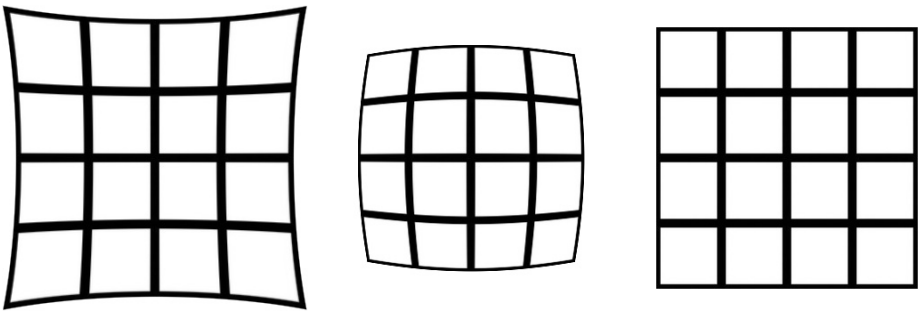

Figure 2.3: Types of radial distortion. Pincushion distortion (*left*), barrel distortion (*middle*), undistorted (*right*).

coordinates $(u,\ v)$ via the relationship

$$u = f_x \frac{X}{Z} + c_x, \quad v = f_y \frac{Y}{Z} + c_y, \tag{2.13}$$

where f_x and f_y are the horizontal and vertical focal lengths in pixel units, respectively. Note f_x and f_y may differ due to non-square pixel sizes. Also, c_x and c_y denote the pixel location of the principal point (ideally in center of image). Equation (2.13) can be equivalently expressed as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = KP = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix},$$

where λ is a scale factor, and K is known as the camera intrinsic matrix.

Visual Odometry

Visual odometry is the algorithmic process of analyzing a chronological sequence of images from one or more cameras to estimate the pose of the camera or cameras. The general VO pipeline is shown in Figure 2.4.

In **feature-based** VO, pixel locations (i.e. features) corresponding to corners within the current image are detected using an algorithm such as FAST [6]. A feature descriptor such as ORB [7] describes (numerically) the local appearance around the feature point and is ideally invariant to changes in lighting, translation, scale, and in-plane rotation. Features in the current and previous image can then be matched by comparing feature descriptors

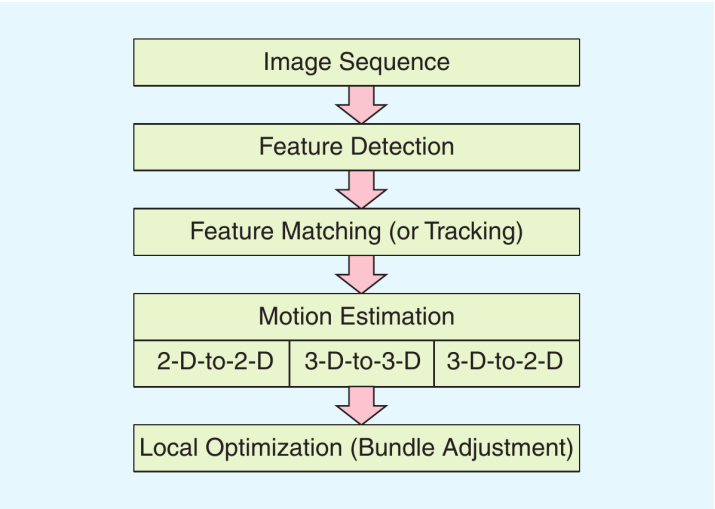

Figure 2.4: A feature-based visual odometry pipeline [5].

(using the Hamming distance is common). After feature detection and matching occurs, the relative change in pose of the camera system from the previous to the current image is estimated. By concatenating the relative poses between images, the position and orientation of complete trajectory can be estimated. Depending on the type of sensor used, this process uses different types of methods.

As shown in figure 2.4, the motion estimation step contains three different options for computing relative pose. The **2D-to-2D** method is more common with monocular cameras, in which motion is estimated from 2D feature correspondences by utilizing the epipolar constraint [5]. This method can be used to estimate 3D points (up to a scale factor) corresponding to the 2D features. The **3D-to-3D** method computes motion from 3D structure correspondences (i.e. 3D points) which is possible when using a stereo

(binocular) camera setup. The stereo camera system has two cameras, allowing for triangulation of 3D points at each time instant. Finally the **3D-to-2D** method can be employed by both mono or stereo cameras. In the monocular case, 3D points need to first be determined using the 2D-to-2D method, then the 3D-to-2D method can be used. More detailed review of these motion estimation methods can be found in [5]. Optionally, **local bundle adjustment** (a least-squares optimization) can be performed over multiple previous pose estimates, minimizing the image reprojection error of the estimated 3D structure points in order to improve trajectory accuracy.

In contrast to the more "classical" feature-based VO, also known as indirect VO, **direct** VO consists of directly utilizing raw pixel intensities to include a larger percentage of available image information. Direct methods minimize the photometric error between images in order to estimate transformations between camera poses. These methods tend to have improved accuracy over indirect (feature-based) methods due to their utilization of more available information. However, direct methods also incur a larger computational burden than indirect. Many VO algorithms attempt to achieve the accuracy of direct methods with the computational efficiency of indirect methods by combining the two into hybrid direct/indirect VO. In fact, the VIO algorithm used in this thesis is a hybrid method [8].

Visual-Inertial Odometry

The accuracy and robustness of VO can be greatly improved upon by using an inertial measurement unit (IMU), which measures linear acceleration and angular rate. Using a camera setup with a rigidly attached IMU to perform locally accurate state estimation is known as visual-inertial odometry (VIO). Due to the higher refresh rate of IMUs compared to cameras, VIO algorithms can handle rapid movements better than VO algorithms. As such, VIO is a natural choice for fast and dynamic robotic applications such as quadcopter flight. Additionally, the use of an IMU allows for more accurate velocity estimation, which is helpful for precise quadcopter control.

When combining information obtained from a camera and IMU for VIO, there are two main strategies: loose and tight coupling. In **loosely coupled** VIO systems, state is estimated independently between the camera and IMU. This means that VO and IMU integration are performed separately, then combined later to obtain a fused estimate. In contrast, **tightly coupled** VIO systems combine visual information (2D features, pixel intensity) and IMU information (acceleration, angular velocity) to obtain the final estimate. Loose coupling, while computationally efficient, results in loss of information. Tight coupling achieves higher accuracy within a single process. In figure 2.5, a comparison of loose and tight VIO frameworks is shown.

VIO algorithms can be further bisected in groupings of filtering-based or optimization-based methods. In **filtering-based** methods, the current state estimate only depends on the current measurement of the system

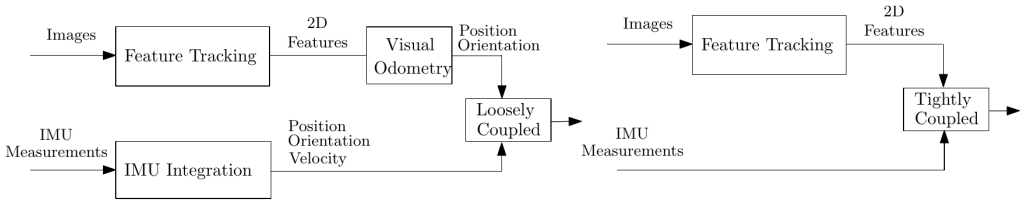

Figure 2.5: Loosely coupled (*left*) and tightly coupled (*right*) feature-based VIO paradigms [9].

and the previous state estimate. As such, filtering based methods are computationally efficient, but also will inherit inaccurate estimates in the current state estimate, effectively "locking" any poor measurement in the filter [9]. It is also well known that the extended Kalman filter (a popular choice for VIO methods) for nonlinear systems is not optimal in the least-squares sense, which may cause inconsistencies due to linearization error.

Alternatively, **optimization-based** VIO aims to improve state estimates along the trajectory history of the robot. This is in contrast to filtering-based VIO which only provides an estimate of the current state without updating previous ones. Since optimization approaches relinearize part (or all) of the previous measurements as the estimate is updated, they are generally more accurate than filtering [9]. Early research on VIO algorithms tend to favor filtering-based methods for their computational efficiency. More recently, optimization-based methods have become increasingly popular due to their better accuracy and availability of more powerful computers.

VIO of Choice

Given the measurements and comparisons of different VIO algorithms provided by [10] as described in the next chapter, the VIO algorithm we chose to use for this thesis was ROVIO [8], due to its balance of performance and computational efficiency. ROVIO is semi-direct, filtering-based, and uses tight coupling between inertial and visual measurements.

3 RELATED WORKS

In this chapter we review multiple areas of research related to autonomous vision-based quadcopter landing. We will discuss VIO, FOV-constrained quadcopter flight, perception-aware quadcopter control and landing schemes for quadcopters.

3.1 Visual-Inertial Odometry

Here we provide a review of multiple VIO frameworks, considering both the filtering and optimization methods used for visual-inertial motion estimation. For a more in-depth review, [11] provides a survey of visual-inertial navigation, with notes on observability analysis and open questions in the field. Also, [9] reviews multiple methods of VIO for use with quadcopters.

In [12] the authors present a Multi-Sensor-Fusion EKF method that can process delayed, relative and absolute measurements from many different sensors and sensor types. While this work is general to multiple types of sensors, it can be used with visual odometry as well. This method fuses sensors in a loosely coupled manner, such that the computational burden is lower than tightly-coupled methods, but does not exploit correlations among all measurements. Iterative EKF is used, which linearizes the measurement model iteratively, providing better performance than a standard EKF.

In [13], a tightly coupled sensor fusion method is proposed. Specifically,

this method fuses global positional information with visual and inertial measurements in a tightly-coupled nonlinear optimization–based estimator. A cost function including visual, inertial and global measurement errors is used to perform an optimization that estimates the state.

In [14], authors use VIO to estimate state then use monocular dense mapping to reconstruct the surrounding environment. This map can then be used to perform trajectory planning. A rapidly exploring random graph is used to search an asymptotically optimal path.

In [15], a RGB-D sensor based system is used to stabilize and control a small quadrotor. Similar to the previous mentioned work [14], this work estimates its own position using the RGB-D sensor fused with an IMU, builds a dense 3D model of the environment, and uses this model to plan trajectories through the environment. A PID position controller is used to control the position of the robot. It should be noted that a 3D map is not necessary for stabilization of the quadcopter, but is needed for obstacle avoidance and trajectory planning.

Authors in [10] performed benchmark comparisons of various VIO methods on different computing devices with applicability for flying robots. The VIO algorithms they test are outlined here, and shown in Table 3.1

MSCKF: The multi-state constraint Kalman filter algorithm [16] proposes a measurement model that expresses the geometric constraints between all of the camera poses that observed a particular image feature, without the need to maintain an estimate of the 3D feature position in the state.

Algorithm	Coupling	Backend	Frontend
MSCKF [16]	Tight	EKF	Indirect
OKVIS [17]	Tight	Sliding window opt.	Indirect
ROVIO [8]	Tight	Iterated EKF	Semi-direct
VINS-Mono [18]	Tight	Sliding window opt.	Indirect
SVO+MSF [19]	Loose	EKF	Semi-direct
SVO+GTSAM [20]	Tight	Factor graph opt.	Semi-direct

Table 3.1: Summary of various VIO algorithms.

Performance: Accuracy was consistent regardless of the platform. In addition to robustness, it generally provided modest resource usage and low per-frame processing time. However, most of the modern algorithms (below) are able to achieve higher overall accuracy with a manageable increase in resource requirements.

OKVIS: Open keyframe-based visual-inertial SLAM [17] uses non-linear optimization on a sliding window of keyframe poses. The cost function includes visual landmark reprojection errors, and inertial errors. BRISK descriptors are computed from detected corners to feature matching between frames.

Performance: Demonstrated accurate performance across all of the hardware platforms, including the embedded systems, despite low update rates due to long per-frame processing times.

ROVIO: Robust visual inertial odometry [8] is an EKF-based method. In addition to FAST corner features, whose 3D positions are parameterized with

robot-centric bearing vectors and distances, multi-level patches are extracted from the image around the features. The patch features are tracked, warped based on IMU-predicted motion, and the photometric errors are directly used in the update step as innovation terms to produce a tightly coupled framework.

Performance: Performance was accurate and consistent, suggesting good robustness to challenging trajectories, given a sufficiently powerful computer.

VINS-Mono: VINS-Mono [18] is a nonlinear optimization-based sliding window estimator, tracking robust corner features, similar to OKVIS. IMU measurements are pre-integrated before being used in the optimization, and a tightly-coupled procedure for relocalization is proposed.

Performance: The most consistently accurate and robust across all of the hardware platforms. Superior performance comes at the cost of a potentially prohibitive computation expenditure.

SVO+MSF: Multi-Sensor Fusion (MSF) [12] is a general EKF framework for fusing data from different sensors in a state estimate. Semi-Direct Visual Odometry (SVO) [21] is a computationally lightweight visual odometry algorithm that aligns images by tracking FAST corner features [6] and minimizing the photometric error of patches around them. The outputs from both frameworks are fused [19]. This loose coupling can cause pose scale issues.

Performance: Highest level of computational efficiency, but with the corresponding lowest level of accuracy.

SVO+GTSAM: SVO is combined with the Georgia Tech Smoothing and Mapping (GTSAM) toolbox [20]. The same visual odometry frontend as in the SVO+MSF system has also been paired with a full-smoothing back-end performing online factor graph optimization using iSAM2 [22].

Performance: Not as robust as other methods, but produces the most accurate trajectories for many of the platform-dataset combinations, when considering the algorithms without loop closure.

3.2 Quadcopter Field-of-View Constraints

The following works contain relevant discussions regarding maintaining FOV constraints for quadcopter systems.

In [23], the problem of aggressive collision avoidance in an unknown environment with limited FOV sensing is addressed. In this work, they consider the scenario in which initially the quadcopter is able to sense the obstacles in front of it (assuming a front-facing camera is used). However, once the quadcopter performs a high-acceleration maneuver to fly between obstacles, the camera may no longer see the obstacles. This work tackles this problem by contributing a framework that allows *safe* execution of high-acceleration motion primitives, by choosing a safe motion primitive that was generated in the past when obstacles were observed. An MPC

framework was used to derive the necessary conditions for safety. While providing a solution for a stationary environment, this framework could struggle in dynamic environments when the obstacles are not in the camera FOV.

Authors in [24] consider the problem of keeping a landmark in the FOV of a down-facing camera that is attached to a quadcopter, while navigating closer to the landmark. As shown in Figure 3.1, the problem lies in the fact that the quadcopter needs to tilt toward the landmark, which can cause the down-facing camera to lose view of the landmark. To address this issue, the quadcopter needs to both fly toward the landmark and increase altitude, which effectively allows the FOV to capture more of the scene. In [24], authors parameterize the trajectory using B-splines, and exploit the differential flatness of the quadcopter system to solve a sequential quadratic program (SQP) to optimize the trajectory, which is non-convex problem. While showing promising performance in simulation, the result was not verified in hardware. Similarly, [25] addressed FOV-constrained target tracking while avoiding obstacles and occlusions using similar SQP techniques.

In [26], authors contribute a time-optimal algorithm for efficient path parameterization for quadcopters with FOV constraints. This approach assumes an *a-priori* path is provided, and the task of the authors is to find the shortest time such that no FOV constraints are violated. They consider the control of a quadcopter that has a front-facing camera, and are able to

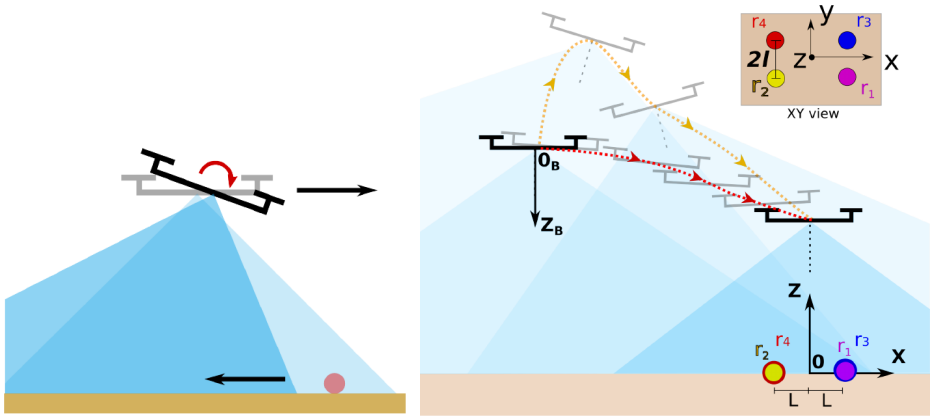

Figure 3.1: *Left:* The quadcopter needs to pitch to the right to move toward the red target, causing the target to leave the camera FOV. *Right:* By moving upward and to the right, the quadcopter can keep the targets in the FOV (yellow dashed line) with a faster completion time than a near-hover method (red dashed line). Figures from [24].

obtain a convex constraint that can be used to solve this problem. This approach is useful for drone racing in which a path may already be planned, and the quadcopter controller needs to determine how fast the fly. However, for more general FOV-constrained applications such as tracking and landing, the planning and perception contrainst need to be considered at the same time.

In [27], the authors consider image-based visual servoing (IBVS) control of a quadcopter, while also guaranteeing visibility constraints with a control barrier function (CBF) approach. IBVS control does not involve estimation of the pose of the quadrotor, and instead provides control based on the error between current and desired features on the image plane (tracking in image space, not state space). In this way, the dynamics of the image features

are derived from the quadrotor dynamics and the camera projection. A virtual camera approach is adopted to achieve a simplified image feature dynamics and facilitate the controller design, and also proves Lyapunov stability of the specific IBVS implementation. Unfortunately the IBVS method is completely dependant on viewing the image features to work properly, forcing the quadcopter to remain in near-hover to make experiments successful. While the near-hover condition is maintainable for slow flight, fast quadcopter flight violates this condition considerably.

3.3 Perception-Aware Quadcopter Control

Closely related to FOV-constrained control is perception aware (PA) control. In general, PA control aims to control an agent such that some perception goals are met. For example goals could include soft/hard FOV constraints (previous section only considers hard constraints) and viewing visually feature-rich areas to improve VIO accuracy. The following works look into various PA methods designed specifically for quadcopter/multicopter flight.

Perception-aware model predictive control (PAMPC) provides an approach that adds a perception objective into their MPC framework [1]. They model the perception cost as minimizing 1) position of a point of interest (POI) from the center of the image plane and 2) the velocity of the POI in the image frame. A 3D POI in the world frame is translated via quaternions to the point of interest in the camera frame, which is then projected onto the

image plane using the pinhole camera model. The resulting MPC is a non-linear program with quadratic costs. This is approximated by a sequential quadratic program (SQP) where the solution is iteratively approximated and used as a MPC. This PAMPC method can be considered as implementing a soft FOV constraint, since the FOV term is in objective function of the optimization. Compared to [24], PAMPC does not restrict the resulting trajectory to be a piecewise polynomial, allowing for increased dynamic feasibility of the resulting path. While PAMPC was motivated for keeping salient image features in the view, it can be used for keeping any 3D point in the FOV. PAMPC is tested in hardware on a quadcopter using the ACADO toolkit [28] for MPC.

In [29] aggressive quadcopter speeds are achieved using a PA motion planning algorithm. Specifically, differential flatness is used to design the position trajectory of the quadcopter, while the yaw trajectory is determined by ensuring better observation of visual features. The method is considered for a quadcopter passing through specified waypoints. Compared with PAMPC, [29] maximizes the number of features visible in consecutive keyframes as opposed to enforcing that a certain set of features stay in view. An initial optimization minimizes the total flight time by finding position polynomials that satisfy given waypoints. A second optimization occurs at each time step of the trajectory, and minimizes the yaw acceleration as well as the relaxed visibility function. This algorithm converges to a local optimum due to non-convexity.

In [30], a MPC framework similar to PAMPC is proposed for multirotor flight. The main difference is that the camera used is on an gimbal such that the camera can freely view its surrounding area without moving the multirotor body. In general, a solution like this one is a better idea for larger drones that can hold such a gimbal. However, for smaller more agile quadcopters, it may be difficult to use a gimbal method due to weight or size restrictions. If the camera on the gimbal is used for a VIO pipeline, another source of potential error (i.e. measuring the gimbal inverse kinematics) is introduced.

In [31] PANTHER is introduced, a real-time PA trajectory planner for multirotors. This work plans trajectories to keep dynamic obstacles in the camera FOV while simultaneously avoiding them. Similarly, authors in [32] propose RAPTOR, a robust and PA trajectory replanner for fast quadcopter flight. Like in PANTHER, they propose a PA planning method to actively observe and avoid unknown obstacles. From these two works it is apparent that PA-based flight has much promise for use in quadcopter obstacle avoidance.

3.4 Quadcopter Landing Schemes

In this final related works section, landing algorithms and control schemes are discussed, with applications varying from landing on moving objects to landing while counteracting the ground effect. First, we'll mention some

past and present fiducial marker systems that have been used for drone landing.

In 2002 the "H" shape landing pad [33] was used for an autonomous helicopter landing task. Similarly the "T" shape fiducial marker was used in [34]. More advanced fiducial markers also have been developed including recursive circular marker sets [35] and the first and second generations of AprilTag [36, 37], which boast a low false-detection rate, and resemble a grid of black and white squares. In [4] authors show the AprilTag 3 algorithm is faster and has higher recall than both AprilTag 2 and the popular ArUco detectors. Various classical computer vision techniques such as Hu moment invariants, Hough transform, Gaussian smoothing and Canny edge detection can be used to process the UAV landing pad images to determine altitude and attitude. Metrics used to compare such methods consist of estimation rate, success rate, attitude accuracy and positioning accuracy [38].

In [39], the authors consider a cooperative robotic system between a UAV and unmanned ground vehicle (UGV). They contribute an autonomous landing system for charging of the UAV such that the cooperative system can perform long missions. The scheme utilizes fiducial markers for the UAV to track the UGV, and a velocity controller that combines control barrier function (CBF) and control Lyapunov function (CLF) constraints into a QP. However [39] does not consider quadcopter attitude as a potential cause for loss of view of the landing pad, since the method requires a small attitude assumption.

In [40], authors consider the quadcopter landing problem for a moving target. A disturbance observer-based control method is used for robustness against external disturbances during low altitude flight. To avoid target loss situations a heuristic angle scheduling algorithm is proposed. An IR beacon is placed on the landing pad, and an IR camera on the quadcopter tracks it. Similarly, authors of [41] perform a landing of a quadcopter on a car moving at 15km/hr. They use an MPC for trajectory generation with a nonlinear feedback controller for tracking. While this thesis does not test with a moving landing pad (but is an area for future investigation), we believe our FOV-constrained approach would generalize well to such a case, due to its ability to prioritize visibility of the landing pad.

Authors in [42] propose an integral sliding mode altitude controller that is used in conjunction with ground effect compensation for landing of a model helicopter. The integral sliding mode allows for asymptotic convergence to the desired height. Similarly, [43] also employs classical control methods for lading of a quadcopter. Specifically, a nominal PID controller is combined with a robust compensator and ground effect compensator to provide better low-altitude tracking.

Authors in [44] provide a neural network (NN) based approach for the stable quadcopter landing problem. They present Neural-Lander, a deep learning base nonlinear controller with guaranteed stability for landing. The NN approach has potential benefits over the more classical control methods by being able to learn unmodeled dynamic nonlinearities.

4 FIELD-OF-VIEW CONSTRAINED LANDING

This chapter provides the main contributions of this work. First, FOV constraints linear in acceleration are derived such that they can be included in a min-snap QP problem. Then, the search and landing pipeline is discussed, which contains tracking the search trajectory, checking landing feasibility and generating then tracking the landing trajectory.

4.1 FOV Constraints for Min-snap QP

It is common in vision-based landing research to assume the UAV quadcopter maintains small roll/pitch angles (e.g. [39]). However, this assumption can be easily violated in real-world flight and can cause the landing pad to leave the FOV of the quadcopter camera. The consequences of this could include slow landing or complete landing failure in which the quadcopter cannot re-attain FOV of the landing site. Our approach provides FOV constraints to ensure visibility of the landing pad from the down-facing camera with no assumption of small roll/pitch angles.

Overview of Approach

To support high-visibility of the landing pad, we propose linear constraints to include in the minimum-snap QP (2.6) to satisfy the FOV requirement. Given a xz-planar quadcopter model (see Figure 4.1) we describe the FOV

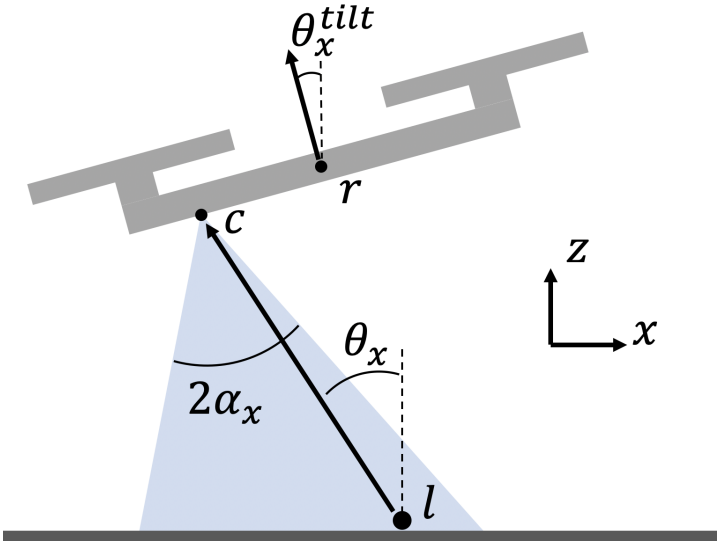

Figure 4.1: FOV constraint geometry of a planar quadcopter. Note that $\boldsymbol{r}$ is the quadcopter position, $\boldsymbol{c}$ is the down-facing camera position, $\boldsymbol{l}$ is the landmark position to keep in the FOV. Also, θ_x^{tilt} is the tilt angle toward the global x-direction, α_x is half the vertical FOV angle, and θ_x is the angle relating the landmark-to-camera vector with the global z axis.

constraint in terms of the quadcopter tilt angle θ_x^{tilt}. For the constraint to be usable in the QP, it must be linear in terms of the decision variables. As such we replace the tilt angle with an equivalent expression using quadcopter acceleration (i.e. $\arctan(\frac{a_x}{a_z+g})$). Noting $\tan(x)$ is monotonic on $x \in (-\pi/2, \pi/2)$, we obtain two constraints linear in acceleration (4.5), (4.6). We then perform the same steps for the yz-planar quadcopter model.

Derivation of Constraints

For the planar (xz plane) quadcopter in Figure 4.1 with a mass-normalized thrust τ, the acceleration of the planar quadcopter is

$$a_x = \tau \sin(-\theta_x^{tilt}) \tag{4.1}$$

$$a_z = \tau \cos(\theta_x^{tilt}) - g. \tag{4.2}$$

In order to keep the landmark $\boldsymbol{l} = \begin{bmatrix} l_x & l_y & l_z \end{bmatrix}^T$ in the camera FOV (with vertical FOV angle $2\alpha_x$), θ_x^{tilt} must be constrained such that

$$-(\alpha_x - |\theta_x|) \leq \theta_x^{tilt} \leq \alpha_x - |\theta_x|, \tag{4.3}$$

where $\theta_x = \tan^{-1}(\frac{l_x - c_x}{c_z - l_z})$ and $\boldsymbol{c} = \begin{bmatrix} c_x & c_y & c_z \end{bmatrix}^T$ denotes the camera position. Combining constraint (4.3) with (4.1) and (4.2), we obtain linear constraints on the acceleration:

$$\begin{aligned} -\tan(\alpha_x - |\theta_x(t_i)|) \leq \frac{-a_x}{a_z + g} &\leq \tan(\alpha_x - |\theta_x(t_i)|) \\ \implies \qquad |a_x| &\leq \tan(\alpha_x - |\theta_x(t_i)|)(a_z + g), \end{aligned} \tag{4.4}$$

where the final implication is allowed by assuming $a_z + g > 0$, and where $\theta_x(t_i)$ is the θ_x value at time t_i. Explicitly choosing the θ_x values at different time points allows this constraint to be linear. In practice $\theta_x(t_i)$ values are chosen as $m + 1$ equally spaced values from $\theta_x(t_0)$ to $\theta_x(t_m)$ (same process

for $\theta_y(t_i)$). Re-arranging constraint (4.4) we can obtain constraints linear in acceleration:

$$a_x - a_z \tan(\alpha_x - |\theta_x(t_i)|) \leq g \tan(\alpha_x - |\theta_x(t_i)|), \tag{4.5}$$

$$-a_x - a_z \tan(\alpha_x - |\theta_x(t_i)|) \leq g \tan(\alpha_x - |\theta_x(t_i)|). \tag{4.6}$$

As can be seen by constraint (4.4), we need to ensure $\tan(\alpha_x - |\theta_x(t_i)|)(a_z + g) \geq 0$ for feasibility of the QP. A weak (i.e. trivial to satisfy) assumption we can make is $a_z + g \geq 0$, meaning the quadcopter should have less than a $\frac{\pi}{2}$ tilt angle. However we cannot easily assume $\tan(\alpha_x - |\theta_x(t_i)|) \geq 0,\ \forall t$. In practice we check $\alpha_x - |\theta_x(t_0)| \geq 0$ is satisfied before attempting to solve the QP, since $\alpha_x - |\theta_x(t)| \geq 0 \implies \tan(\alpha_x - |\theta_x(t)|) \geq 0$, and $|\theta_x(t_0)| \geq |\theta_x(t_i)|$, $\forall i \in \{1, \ldots, m\}$.

Practical Considerations

Planar to 3D: To extend the planar case, we create a constraint analogous to (4.4) for the yz plane. Note this may cause small FOV violations if the quadcopter yaw value is not close to 0 or π. This may be addressed in future work, but can currently be naively addressed by decreasing the value of α_x.

From camera trajectory to quadcopter trajectory: Note that technically, using the FOV constraints (4.4) when solving QP (2.6) will provide a trajectory for the camera center. In the case where the camera center and

quadcopter center do not not align, this could pose an issue when trying to control the yaw of the quadcopter. Specifically, yawing the quadcopter would move the camera position, which could violate the desired FOV constraints. To fix this, we can use QP (2.6) to determine the camera trajectory and quadcopter yaw, then analytically determine the quadcopter trajectory afterwards.

4.2 Search and Landing Pipeline

In this section, a step-by-step description of the FOV-constrained precise landing pipeline is provided. We discuss tracking the search trajectory, checking landing feasibility, generating the FOV-constrained landing trajectory with a min-snap QP, and tracking the trajectory using PAMPC. Figure 4.2 provides a graphical overview of the flight and landing process.

Searching for the AprilTag Marker

Upon take-off of the quadcopter, a pre-generated search trajectory is loaded and tracked by the quadcopter. We use MPC to track this trajectory, but any tracking controller will work. Once the AprilTag marker comes into the FOV of the down-facing camera, the AprilTag algorithm is initiated.

The output of the AprilTag algorithm is the pose of the AprilTag marker from the coordinate frame of the down-facing camera (see Figure 4.3). This relative pose can be represented as transformation matrix $T_A^C \in \mathbb{R}^{4\times4}$.

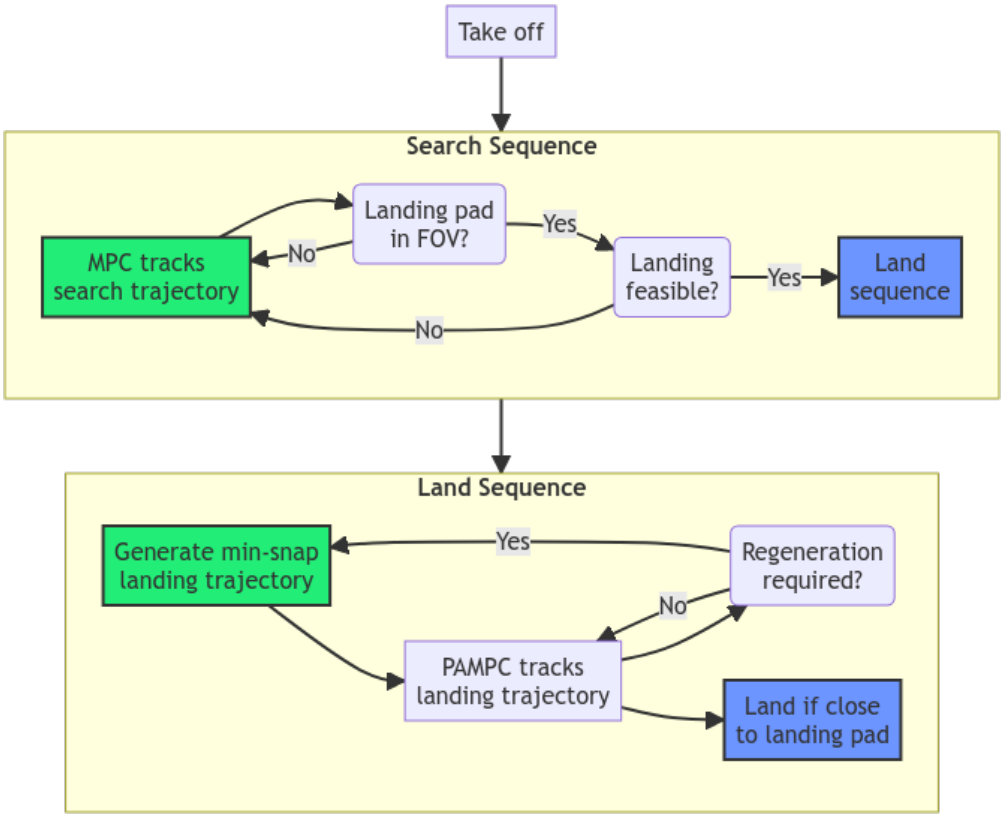

Figure 4.2: Quadcopter flight and landing process. The process starts with *Take off*. Each sequence block starts with a green block, and finishes with a blue block.

However, we need to compute T_A^W, the pose of the AprilTag marker expressed in the VIO world frame, in order to generate a trajectory to the landing pad.

After obtaining T_A^C, we can determine T_A^W with the following expression:

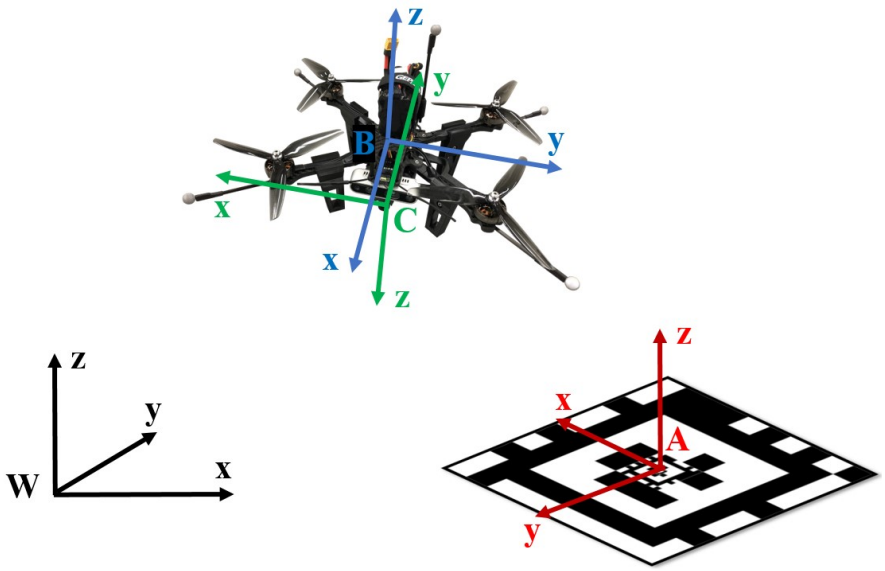

Figure 4.3: Coordinate frames include the world frame W, quadcopter body frame B, down-facing camera frame C and AprilTag frame A.

$$T_A^W = T_A^C T_C^B T_B^W, \tag{4.7}$$

where

T_A^W: AprilTag frame expressed in VIO world frame

T_A^C: AprilTag frame expressed in down-facing camera frame

T_C^B: Down-facing camera frame expressed in quadcopter body frame

T_B^W: Quadcopter body frame expressed in VIO world frame

are known transformation matrices obtained from the AprilTag algorithm (T_A^C), measurement of hardware (T_C^B) and VIO (T_B^W). Note that exponential

smoothing is used to reduce noise of the AprilTag pose estimate. This smoothing works well for a stationary landing pad, but a Kalman filter would likely need to be implemented if a moving landing pad is considered.

Once T_A^W has been computed, the quadcopter needs to check feasibility for solving a min-snap QP.

Checking Landing Feasibility

As mentioned in the previous section, before we generate the landing trajectory, we first need to check feasibility of the FOV constraints. Noting that $\theta_x(t_m) = \theta_y(t_m) = 0$ for landing in the center on the landing pad, we just have to check the following is satisfied:

$$\alpha_x - |\theta_x(t_0)| > 0, \tag{4.8}$$

$$\alpha_y - |\theta_y(t_0)| > 0, \tag{4.9}$$

where $\theta_x(t_0)$ and $\theta_y(t_0)$ can be computed using $\theta_x = \tan^{-1}(\frac{l_x - c_x}{c_z - l_z})$ and $\theta_y = \tan^{-1}(\frac{l_y - c_x}{c_z - l_z})$ respectively. Here, (c_x, c_y, c_z) is the camera position as estimated by VIO, and (l_x, l_y, l_z) is the landing pad position in the VIO world frame as computer by equation (4.7). If the check is successful, we proceed to generating the landing trajectory.

Landing Trajectory Generation

The FOV-constrained min-snap QP for generating the landing trajectory extends the original min-snap QP (2.6) as follows:

$$\min_{\substack{\boldsymbol{r}_{T_{ij}},\ \psi_{T_{ij}} \\ \forall i \in \{0,\ldots,n\} \\ \forall j \in \{1,\ldots,m\}}} \quad \int_{t_0}^{t_m} \mu_r \left\| \mathbf{r}_T^{(k_r)}(t) \right\|^2 + \mu_\psi \left(\psi_T^{(k_\psi)}(t) \right)^2 dt \tag{4.10}$$

$$\text{s.t.} \quad \mathbf{r}_T^{(p)}(t_j) = \mathbf{r}_j^{(p)} \text{ or free}, \quad j = \{0, m\};\ p = 0, \ldots, k_r \tag{4.11}$$

$$\psi_T^{(p)}(t_j) = \psi_j^{(p)} \text{ or free}, \quad j = \{0, m\};\ p = 0, \ldots, k_\psi \tag{4.12}$$

$$\sum_{i=p}^{n} (\boldsymbol{r}_{T_{ij}} - \boldsymbol{r}_{T_{i,j+1}}) = 0, \quad j = 1, \ldots, m-1;\ p = 0, \ldots, k_r \tag{4.13}$$

$$\sum_{i=p}^{n} (\psi_{T_{ij}} - \psi_{T_{i,j+1}}) = 0, \quad j = 1, \ldots, m-1;\ p = 0, \ldots, k_\psi \tag{4.14}$$

$$|a_x(t_j)| \leq \tan(\alpha_x - |\theta_x(t_j)|)(a_z(t_j) + g), \quad j = 1, \ldots, m-1 \tag{4.15}$$

$$|a_y(t_j)| \leq \tan(\alpha_y - |\theta_y(t_j)|)(a_z(t_j) + g), \quad j = 1, \ldots, m-1 \tag{4.16}$$

where μ_r and μ_ψ make the integrand nondimensional and $[a_x(t), a_y(t), a_z(t)]^T := \boldsymbol{r}_T^{(2)}(t)$. Boundary conditions for position and yaw and their derivatives are encoded in (4.11) and (4.12) respectively. Continuity of the piecewise polynomials and their derivatives is ensured by (4.13) and (4.14). FOV constraints are encoded in (4.15) and (4.16).

Boundary Conditions: Note initial conditions (i.e. $\mathbf{r}_0^{(p)}$ and $\psi_0^{(p)}$) are given by the current quadcopter state, while final conditions (i.e. $\mathbf{r}_m^{(p)}$ and

$\psi_m^{(p)}$) are given by the desired quadcopter landing state. Since the quadcopter will land on the AprilTag marker, position and yaw final conditions are given by T_A^W (i.e. the AprilTag pose estimate in the world frame).

Result of the QP: The QP minimizes of the piecewise polynomial coefficients from (2.4). As such the QP returns these coefficients, which can easily be used to find $\mathbf{r}_T(t)^{(p)}$ and $\psi_T(t)^{(p)}$ for any time $t \in [t_0, t_m]$ by using (2.4). In other words, the QP allows us to analytically evaluate the position, yaw and the corresponding derivatives (e.g. velocity, acceleration) of the trajectory. A single entry of the full trajectory can be expressed as

$$\begin{bmatrix} x & y & z & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\psi} & \ddot{x} & \ddot{y} & \ddot{z} & \ddot{\psi} & \dddot{x} & \dddot{y} & \dddot{z} & \dddot{\psi} \end{bmatrix}. \tag{4.17}$$

Since the quadcopter is a differentially flat system (see [3]) with flat outputs $[x,\ y,\ z,\ \psi]$, entries (4.17) compose a flatness-based trajectory. In order to track this trajectory using the PAMPC controller from [1], the flatness-based trajectory need to be mapped to a position, velocity and quaternion trajectory (required by PAMPC). From [45], for example, we can easily compute this mapping to obtain a quaternion-based trajectory with entries of the form:

$$\begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & q_w & q_x & q_y & q_z \end{bmatrix}. \tag{4.18}$$

Regeneration of the Landing Trajectory Note that the AprilTag marker pose estimation may contain error that must be accounted for. To account for such errors, we repeatedly solve the the minimum snap QP

program as the quadcopter approaches the landing pad. A heuristic was chosen for deciding when to re-solve for a new landing trajectory. Specifically, we found that re-solving for a new trajectory when the quadcopter was roughly half-way through the current trajectory (in time) produced satisfactory results. While trajectory regeneration helps mitigate AprilTag pose estimation error, it could also be useful for when the AprilTag marker is not stationary in the environment. After trajectory generation, PAMPC is used to track the landing trajectory.

PAMPC Tracking Control

In PAMPC [1], the optimization problem (2.3) from the baseline MPC is augmented with an additional perception term in the objective function.

Notation: $\boldsymbol{r}_B^W$ and $\boldsymbol{q}_B^W$ denote the position and orientation of frame B w.r.t. frame W, respectively. The Hamilton product $(\boldsymbol{q} \cdot \boldsymbol{q}')$ denotes a rotation by $\boldsymbol{q}$ then by $\boldsymbol{q}'$. The identity quaternion (i.e. no rotation) can be expressed by $(\boldsymbol{q}^{-1} \cdot \boldsymbol{q})$, where $(\boldsymbol{q}^{-1})^{-1} = \boldsymbol{q}$. Lastly, $\boldsymbol{q} \odot \boldsymbol{r} := Rot(\boldsymbol{q})\boldsymbol{r}$ results in a rotation of vector $\boldsymbol{r}$ by rotation matrix $Rot(\boldsymbol{q})$ (see (2.2)).

We define $\mathbf{s}(t) := [u_A(\boldsymbol{x}(t))\ \ v_A(\boldsymbol{x}(t))]^T$ as the pixel coordinates of the projected AprilTag marker center and $\mathbf{s}_d := [c_x\ \ c_y]^T$ (i.e. principal point) as the desired pixel coordinates. Using the pinhole model (2.13) and coordinate frames from Figure 4.3, the projected AprilTag coordinates $\mathbf{s}(t)$ can be

calculated (t and $\boldsymbol{x}$ dropped to simplify notation):

$$u_A = f_x \frac{[\boldsymbol{r}_A^C]_x}{[\boldsymbol{r}_A^C]_z} + c_x, \quad v_A = f_y \frac{[\boldsymbol{r}_A^C]_y}{[\boldsymbol{r}_A^C]_z} + c_y, \quad \text{where}$$
$$\boldsymbol{r}_A^C = (\boldsymbol{q}_B^W \cdot \boldsymbol{q}_C^B)^{-1} \odot (\boldsymbol{r}_A^W - (\boldsymbol{q}_B^W \odot \boldsymbol{r}_C^B + \boldsymbol{r}_B^W)), \tag{4.19}$$

where $(\boldsymbol{r}_B^W,\ \boldsymbol{q}_B^W)$ is from the quadcopter state, $(\boldsymbol{r}_C^B,\ \boldsymbol{q}_C^B)$ is constant and can be measured and $\boldsymbol{r}_A^W$ is assumed to be constant and is determined from (4.7). If the AprilTag marker were to be moving in the world frame, it would be possible to estimate the velocity of the marker then estimate $\boldsymbol{r}_A^W$ to solve (4.19).

The perception term $(\mathbf{s}(t) - \mathbf{s}_d)$ can now be embedded into the objective function, where we define

$$\boldsymbol{z}_N := \begin{bmatrix} \boldsymbol{x}(N) - \boldsymbol{x}_d(N) \\ \mathbf{s}(N) - \mathbf{s}_d \end{bmatrix}, \quad \boldsymbol{z}(t) := \begin{bmatrix} \boldsymbol{x}(t) - \boldsymbol{x}_d(t) \\ \mathbf{s}(t) - \mathbf{s}_d \\ \boldsymbol{u}(t) \end{bmatrix}.$$

Following the notation used in optimization problem (2.3) where $F(\boldsymbol{x}, \boldsymbol{u})$ is a discretization of (2.1) with time step $\Delta t := t_i - t_{i-1},\ \forall i \in \{1, \ldots, N\}$, the

optimization problem for PAMPC can then be expressed:

$$\min_{\substack{\boldsymbol{x}(t_1),\dots,\boldsymbol{x}(t_N)\\ \mathbf{s}(t_1),\dots,\mathbf{s}(t_N)\\ \boldsymbol{u}(t_0),\dots,\boldsymbol{u}(t_{N-1})}} \quad \boldsymbol{z}_N^T Q_N \boldsymbol{z}_N + \sum_{i=0}^{N-1} \boldsymbol{z}(t_i)^T Q \boldsymbol{z}(t_i) \tag{4.20}$$

$$\begin{aligned}
\text{s.t.} \quad & \boldsymbol{x}(t_0) = \hat{\boldsymbol{x}}(t_0) \\
& \mathbf{s}(t_0) = \hat{\mathbf{s}}(t_0) \\
& \boldsymbol{x}(t_i) = F(\boldsymbol{x}(t_{i-1}), \boldsymbol{u}(t_{i-1})), \quad \forall i \in \{1,\dots,N\} \\
& \mathbf{s}(t_i) = [u_A(\boldsymbol{x}(t_i)),\ v_A(\boldsymbol{x}(t_i))]^T, \quad \forall i \in \{1,\dots,N\} \\
& \boldsymbol{u}(t_{i-1}) \in \mathcal{U}, \quad \forall i \in \{1,\dots,N\}
\end{aligned}$$

where Q and Q_N indicate weight matrices and the set of admissible control inputs is expressed as

$$\mathcal{U} = \left\{ [\tau\ \ \boldsymbol{\omega}^T]^T \in \mathbb{R}^4 \ :\ \tau \in [\tau_{min},\ \tau_{max}], \quad \boldsymbol{\omega} \in [\boldsymbol{\omega}_{min},\ \boldsymbol{\omega}_{max}] \right\}. \tag{4.21}$$

Effectively, by having the PAMPC track a trajectory that *already* satisfies the FOV constraint (previous subsection), we improve robustness for the FOV-constrained landing. Any disturbances toward the end of the flight trajectory will by compensated for in real time by the PAMPC controller. An overview of quadcopter flight and landing process is shown in Figure 4.2.

5 EXPERIMENT

In this chapter the methods and results for simulated and real-world experiments are presented.

5.1 Method

In both simulation and hardware tests we used ROS 1 and the PX4 flight control software stack. While our work is agnostic to the VIO algorithm used, the results we provide were obtained while using ROVIO [8], chosen for its lightweight implementation and robust performance.

Simulation

Setup: The Gazebo simulator with an iris quadcopter drone is used. This simulator provides good general support for ROS and PX4 integration while being able to provide support for multiple simulated cameras, as well as including advanced physics such as ground effect. The default quadcopter was adapted to include a visual-inertial stereo camera (front-facing), and a monocular camera (down-facing). A Gazebo world was designed such that a sufficient amount of visual features were available for the VIO algorithm (see Figure 5.1). Rviz was used to visualize the states of multiple ROS topics such as the estimated poses of the AprilTag marker and the quadcopter, the position history of where the quadcopter had flown, the output image of

Figure 5.1: Gazebo world simulation. Note the AprilTag marker which is used as the landing pad.

the down-facing camera and the predicted trajectory provided by the MPC controller (see Figure 5.2).

Experiment: For the experiment, the AprilTag marker was placed on the ground plane such that it was initially out of view of the quadcopter. The quadcopter then tracks the pre-generated search trajectory using the MPC controller. The search path trajectory maintained a constant height, and engages in a back and forth motion to thoroughly scan the simulation ground plane for the fiducial marker.

After finding the AprilTag and checking landing feasibility, but before generating a landing trajectory, a desired flight duration (i.e. $t_m - t_0$) must

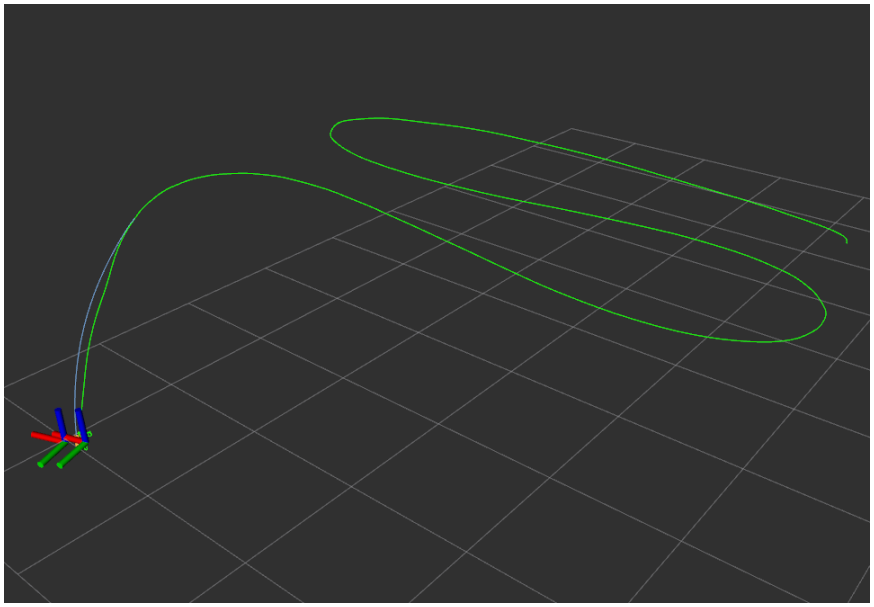

Figure 5.2: The flight path history (green) for a simulation flight displayed in Rviz. The quadcopter first tracks the search trajectory (starts on right). Once the AprilTag is visualized the landing sequence is initialized. The blue trajectory shows the lastest landing trajectory produced by the min-snap QP.

be provided to QP (4.10). This flight duration is heuristically determined based on the distance to the detected AprilTag marker and the current velocity of the quadcopter. A desired average speed is determined given by $s_{avg} = |s_{base}e^{-\mu v_z}|$, where s_{base} is the desired average speed if the quadcopter velocity in the z-direction (i.e. v_z) is zero, and μ is a tuning constant. The flight duration is then computed as $t_{dur} = z/s_{avg}$, where z is height above the landing pad.

Once the quadcopter had followed the current landing trajectory for a time of $0.5t_{dur}$, a new min-snap trajectory is generated ("regeneration"

from figure 4.2). This is done in order to account for errors in the AprilTag location estimation. As the quadcopter continues the landing, the AprilTag will naturally enter the center of the camera FOV, allowing for more accurate landing pad position estimation.

Finally, once the quadcopter descends below a predetermined cutoff height, the motors are shut off and the quadcopter lands on the landing pad. This framework allows for fast and accurate landing.

Hardware

Setup: Before performing experiments, camera calibration and thrust mapping is required. Both the front-facing and down-facing cameras are calibrated using Kalibr. The better the calibration of the down-facing camera is, the better the estimation of the AprilTag position. Regarding thrust mapping, we need to find a mapping from force-normalized thrust to PX4-normalized thrust (i.e. values in the range [0, 1]). To do this, we record log data of a quadcopter flight, then determine the force-normalized thrust at each time step based on the recorded acceleration and attitude. A polynomial is fit between this data and the PX4-normalized thrust that was also recorded at each time step. See figure 5.3 for the quadcopter thrust map.

Experiment: For the hardware experiment, the simulation experiment procedure is repeated, with extra care taken for maintenance of battery voltage to ensure consistent performance. The tests are performed indoors

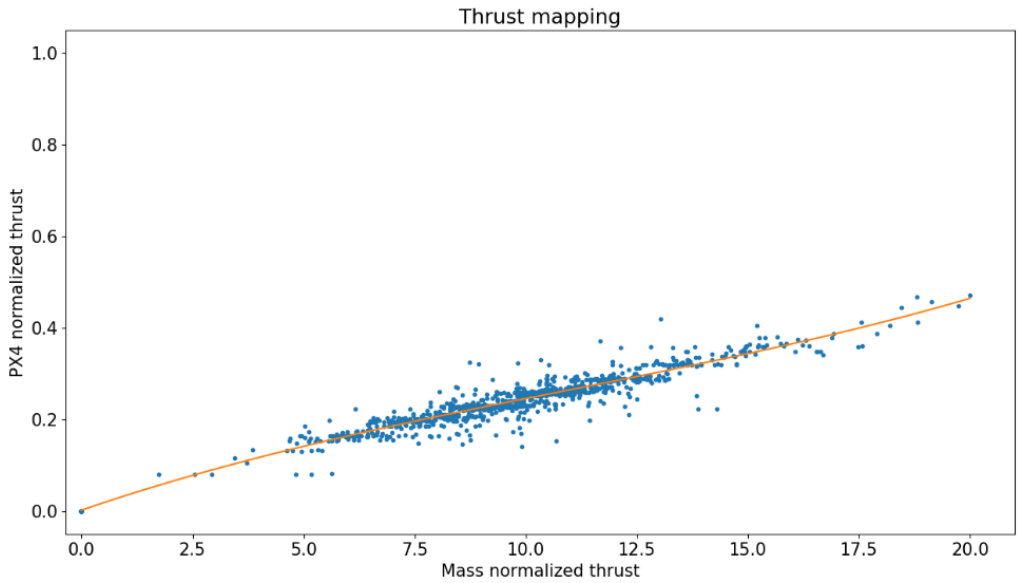

Figure 5.3: A cubic thrust mapping (orange line) fit to PX4 thrust data from a real-world flight.

where a motion capture (mocap) system is used for verification. Mocap can optionally be used to provide a fail safe for the quadcopter if the VIO state estimation fails.

5.2 Results

To investigate the effectiveness of the result, during the flight we record the distance from the projected landing pad center to the center of the camera image plane once the landing pad enters the FOV of the down-facing camera.

From the AprilTag estimation algorithm, the pose of the landing pad w.r.t. the down-facing camera ($\boldsymbol{r}_A^C$) is provided. The point $\boldsymbol{r}_A^C$ in the

camera frame can be projected into the image plane coordinates according the pinhole model. Letting (u_A, v_A) be the horizontal and vertical pixel locations in the image plane of the AprilTag center, we have:

$$u_A = f_x \frac{[\boldsymbol{r}_A^C]_x}{[\boldsymbol{r}_A^C]_z} + c_x, \quad v_A = f_y \frac{[\boldsymbol{r}_A^C]_y}{[\boldsymbol{r}_A^C]_z} + c_y,$$

where f_x, f_y are respectively vertical and horizontal focal lengths in units of pixels and (c_x, c_y) is the image principal point location.

The same experiment is conducted in simulation both with FOV constraints and without FOV constraints. The preliminary results in Figures 5.4 and 5.5 show promising results for the effectiveness of the approach. The tested trajectories show that the FOV constrained flight is able to keep the landing pad center closer to the image principal point, as compared to not using FOV constraints.

Furthermore, while VIO has not yet been used in the hardware setting, an initial test of the landing algorithm was successful while using Mocap for state estimation on the real-world quadcopter. See figure 5.6 for images of the flight and landing.

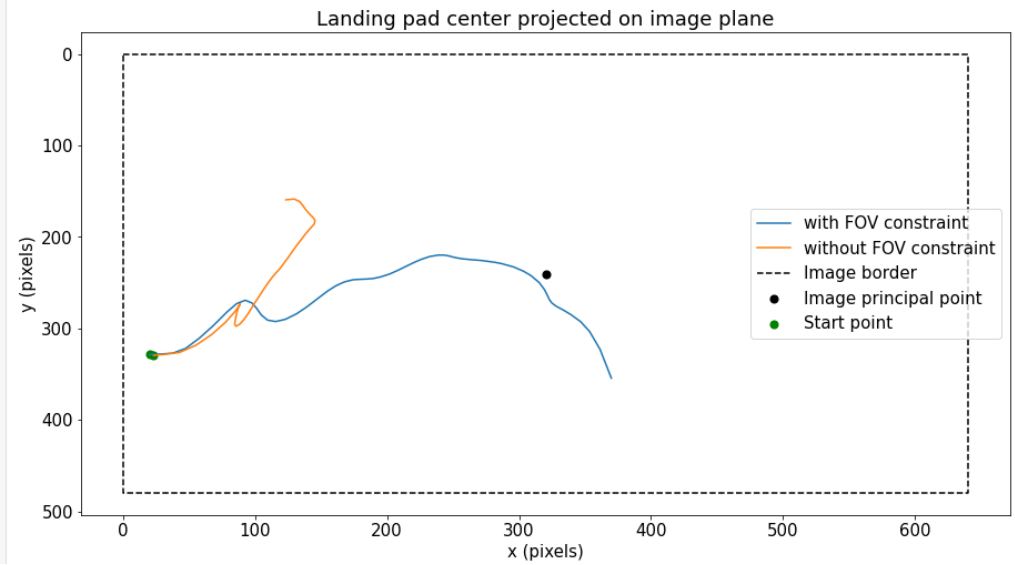

Figure 5.4: Trajectories of the landing pad center projected onto the image plane.

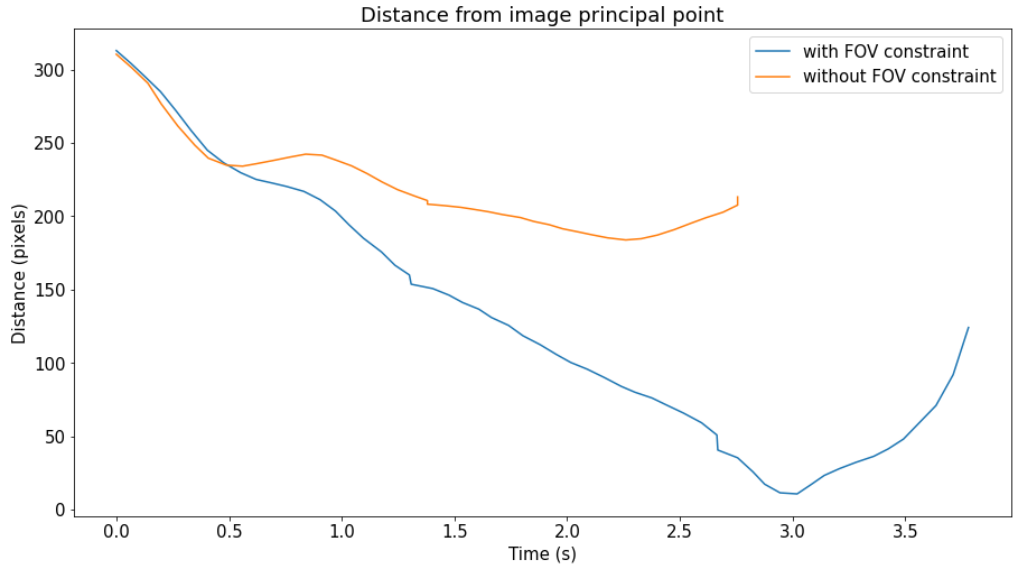

Figure 5.5: Distance between projected landing pad center to image principal point in pixels. Smaller is better.

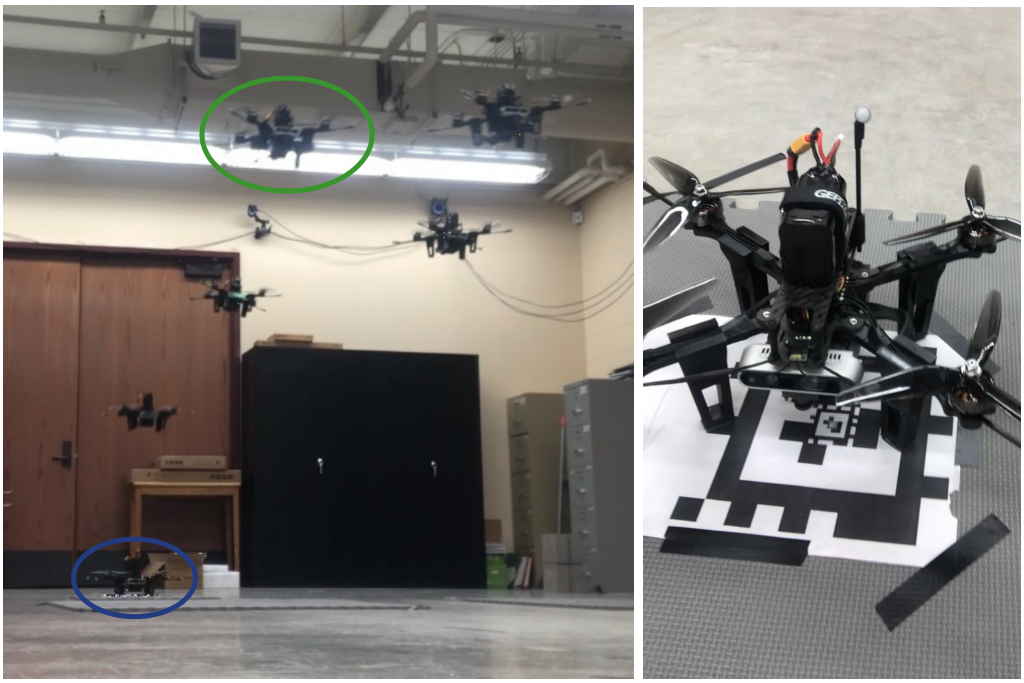

Figure 5.6: Real-world flight and landing of the quadcopter. *Left:* The green oval shows when the quadcopter starts tracking the search trajectory. The blue oval shows a successful landing. *Right:* Close up of the landed quadcopter.

6 CONCLUSION

This work considers the problem of autonomous quadcopter landing for vision-enabled systems. This issue requires a quadcopter-mounted camera to maintain view of the landing area at all times. We tackle this challenge by proposing FOV constraints, linear in the minimum-snap QP decision variables. This enables trajectory generation that inherits the speed of the min-snap QP work, while ensuring visibility of the landing pad for improved landing accuracy. This work utilizes VIO for state estimation, showing high practicality for GPS-denied environments. Use cases for this line of research are extensive and include safer human-robot interaction, more efficient drone delivery services and fast autonomous landing on moving vehicles to name a few.

A QUADCOPTER MATERIALS LIST

Part	Product	Specification
Frame	Pyrodrone Source One	7in
Motors	EMAX ECO II Series 2807	1300KV
Propellors	HQ Prop tri-blade	7in, 4° pitch
Battery	GEPRC Li-ion	3000mAh, 22.2V
FCU	Kakute H7 v2	-
ESC	Tekko 32 4-in-1 ESC	65A
CC	NVIDIA Jetson	Xavier NX
VIO Camera	Intel d435i	stereo + IMU
Mono Camera	Waveshare IMX219-160	160° FOV

Table A.1: Main components of the quadcopter.

BIBLIOGRAPHY

[1] Davide Falanga et al. "PAMPC: Perception-Aware Model Predictive Control for Quadrotors". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2018), pp. 1–8.

[2] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms". In: 2015 IEEE International Conference on Robotics and Automation (ICRA). 2015, pp. 6235–6240.

[3] Daniel Mellinger and Vijay Kumar. "Minimum snap trajectory generation and control for quadrotors". In: 2011 IEEE International Conference on Robotics and Automation. May 2011, pp. 2520–2525.

[4] Maximilian Krogius, Acshi Haggenmiller, and Edwin Olson. "Flexible Layouts for Fiducial Tags". In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, Nov. 2019, pp. 1898–1903.

[5] Davide Scaramuzza and Friedrich Fraundorfer. "Visual Odometry: Part I Tutorial". In: *IEEE Robotics Automation Magazine* (Dec. 2011), pp. 80–92.

[6] Edward Rosten, Reid Porter, and Tom Drummond. "Faster and Better: A Machine Learning Approach to Corner Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (Jan. 2010), pp. 105–119.

[7] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: 2011 International Conference on Computer Vision. Nov. 2011, pp. 2564–2571.

[8] Michael Bloesch et al. "Robust visual inertial odometry using a direct EKF-based approach". In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, Sept. 2015, pp. 298–304.

[9] Davide Scaramuzza and Zichao Zhang. *Visual-Inertial Odometry of Aerial Robots*. Tech. rep. arXiv, June 2019. URL: `http://arxiv.org/abs/1906.03289`.

[10] Jeffrey Delmerico and Davide Scaramuzza. “A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots”. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). May 2018, pp. 2502–2509.

[11] Guoquan Huang. “Visual-Inertial Navigation: A Concise Review”. In: 2019 International Conference on Robotics and Automation (ICRA). May 2019, pp. 9572–9582.

[12] Simon Lynen et al. “A robust and modular multi-sensor fusion approach applied to MAV navigation”. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2013, pp. 3923–3929.

[13] Giovanni Cioffi and Davide Scaramuzza. “Tightly-coupled Fusion of Global Positional Measurements in Optimization-based Visual-Inertial Odometry”. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Oct. 2020, pp. 5089–5095.

[14] Yi Lin et al. “Autonomous aerial navigation using monocular visual-inertial fusion”. In: *Journal of Field Robotics* (2018), pp. 23–51.

[15] Albert S. Huang et al. “Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera”. In: *Robotics Research*. Springer International Publishing, 2017, pp. 235–252.

[16] Anastasios I. Mourikis and Stergios I. Roumeliotis. “A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation”. In: 2007 IEEE International Conference on Robotics and Automation. IEEE, 2007, pp. 3565–3572.

[17] Stefan Leutenegger et al. “Keyframe-based visual–inertial odometry using nonlinear optimization”. In: *The International Journal of Robotics Research* (Mar. 2015), pp. 314–334.

[18] Tong Qin, Peiliang Li, and Shaojie Shen. “VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator”. In: *IEEE Transactions on Robotics* (Aug. 2018), pp. 1004–1020.

[19] Matthias Faessler et al. “Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle: Autonomous, Vision-based Flight and Live Dense 3D Mapping”. In: *Journal of Field Robotics* (June 2016), pp. 431–450.

[20] Christian Forster et al. “On-Manifold Preintegration for Real-Time Visual–Inertial Odometry”. In: *IEEE Transactions on Robotics* (Feb. 2017), pp. 1–21.

[21] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. “SVO: Fast semi-direct monocular visual odometry”. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). May 2014, pp. 15–22.

[22] Michael Kaess et al. “iSAM2: Incremental smoothing and mapping using the Bayes tree”. In: *The International Journal of Robotics Research* (Feb. 2012), pp. 216–235.

[23] Brett T. Lopez and Jonathan P. How. “Aggressive collision avoidance with limited field-of-view sensing”. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, Sept. 2017, pp. 1358–1365.

[24] Bryan Penin et al. “Vision-based minimum-time trajectory generation for a quadrotor UAV”. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, Sept. 2017, pp. 6199–6206.

[25] Bryan Penin, Paolo Robuffo Giordano, and François Chaumette. “Vision-Based Reactive Planning for Aggressive Target Tracking While Avoiding Collisions and Occlusions”. In: *IEEE Robotics and Automation Letters* (Oct. 2018), pp. 3725–3732.

[26] Igor Spasojevic, Varun Murali, and Sertac Karaman. “Perception-aware time optimal path parameterization for quadrotors”. In: (May 28, 2020). arXiv: 2005.13986[cs]. URL: http://arxiv.org/abs/2005.13986.

[27] Dongliang Zheng et al. “Toward Visibility Guaranteed Visual Servoing Control of Quadrotor UAVs”. In: *IEEE/ASME Transactions on Mechatronics* (2019), pp. 1087–1095.

[28] B. Houska, H.J. Ferreau, and M. Diehl. “ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization”. In: *Optimal Control Applications and Methods* (2011), pp. 298–312.

[29] Varun Murali et al. “Perception-aware trajectory generation for aggressive quadrotor flight using differential flatness”. In: 2019 American Control Conference (ACC). IEEE, July 2019, pp. 3936–3943.

[30] Melissa Greeff, Tim Barfoot, and Angela P Schoellig. “A Perception-Aware Flatness-Based Model Predictive Controller for Fast Vision-Based Multirotor Flight”. In: (2020).

[31] Jesus Tordesillas and Jonathan P. How. “PANTHER: Perception-Aware Trajectory Planner in Dynamic Environments”. In: *IEEE Access* (2022), pp. 22662–22677.

[32] Boyu Zhou et al. *RAPTOR: Robust and Perception-aware Trajectory Replanning for Quadrotor Fast Flight*. Tech. rep. arXiv, July 2020.

[33] S. Saripalli, J.F. Montgomery, and G.S. Sukhatme. “Vision-based autonomous landing of an unmanned aerial vehicle”. In: Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292). 2002, 2799–2804 vol.3.

[34] Allen C. Tsai, Peter W. Gibbens, and R. Hugh Stone. “Terminal Phase Vision-Based Target Recognition and 3D Pose Estimation for a Tail-Sitter, Vertical Takeoff and Landing Unmanned Air Vehicle”. In: *Advances in Image and Video Technology*. Springer Berlin Heidelberg, 2006, pp. 672–681.

[35] A. Benini, M. J. Rutherford, and K. P. Valavanis. “Real-time, GPU-based pose estimation of a UAV for autonomous takeoff and landing”. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). May 2016, pp. 3463–3470.

[36] Edwin Olson. “AprilTag: A robust and flexible visual fiducial system”. In: 2011 IEEE International Conference on Robotics and Automation. May 2011, pp. 3400–3407.

[37] John Wang and Edwin Olson. “AprilTag 2: Efficient and robust fiducial detection”. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Oct. 2016, pp. 4193–4198.

[38] Long Xin et al. “Vision-Based Autonomous Landing for the UAV: A Review”. In: *Aerospace* (Nov. 2022), p. 634.

[39] Guanchong Niu et al. “Vision-Based Autonomous Landing for Unmanned Aerial and Ground Vehicles Cooperative Systems”. In: *IEEE Robotics and Automation Letters* (July 2022), pp. 6234–6241.

[40] Nguyen Xuan-Mung et al. “Quadcopter Precision Landing on Moving Targets via Disturbance Observer-Based Controller and Autonomous Landing Planner”. In: *IEEE Access* (2022). Conference Name: IEEE Access, pp. 83580–83590.

[41] Tomas Baca et al. “Autonomous landing on a moving vehicle with an unmanned aerial vehicle”. In: *Journal of Field Robotics* (2019), pp. 874–891.

[42] Kenichiro Nonaka and Hirokazu Sugizaki. “Integral sliding mode altitude control for a small model helicopter with ground effect compensation”. In: 2011 American Control Conference. IEEE, June 2011, pp. 202–207.

[43] Li Danjun et al. “Autonomous landing of quadrotor based on ground effect modelling”. In: 2015 34th Chinese Control Conference (CCC). July 2015, pp. 5647–5652.

[44] Guanya Shi et al. “Neural Lander: Stable Drone Landing Control Using Learned Dynamics”. In: 2019 International Conference on Robotics and Automation (ICRA). May 2019, pp. 9784–9790.

[45] Victor Freire and Xiangru Xu. “Flatness-Based Quadcopter Trajectory Planning and Tracking With Continuous-Time Safety Guarantees”. In: *IEEE Transactions on Control Systems Technology* (2023), pp. 1–16.