Abstract

Fixed-wing Unmanned Aerial Vehicles are extensively used for ground observation as they allow for effective and precise methods of collecting information, but they have a downside as the quality of the information gathered depends on the attitude of the aircraft. There is a coupling between the sensors and the aircrafts attitude states, which is usually decoupled by mounting the sensors to a gimbal which increases the weight of the payload. This paper investigates a control method that minimizes the error caused by the aircraft attitude states when a hyperspectral pushbroom camera is fixed to the aircraft body. This will be done by developing an offline intervalwise Model Predictive Control algorithm that seeks to minimize the distance between the camera centre point and a pre-defined ground path that is to be observed. Since this is an offline method, it is to be run before the flight is initiated. The algorithm is implemented using C++ and the ACADO Toolkit, and the optimized paths will be simulated using a Software-in-the-Loop simulator. The application developed is able to optimize both curved and piecewise linear paths. It is able to optimize linear corners with an angle of 45°, and curved turns with an angle of 90° and also 180°. However, testing proved that the application was not very robust when it comes to optimizing paths containing several sharp turns quickly after another, or long straight paths right after a turn. In addition the optimization is very time consuming. Simulation of the optimized path show that the precision of tracking the path can be done with a mean error of below 5m.

Abstract in Norwegian

Preface

This paper is submitted in partial fulfillment of the requirements for the Master of Science degree at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor Professor Tor Arne Johansen for guidance and help with understanding the problem at hand and possible solutions. A big thank you goes to PhD Candidate Siri Holthe Mathisen for always leaving her office door slightly open, so I could ask questions regarding almost anything whenever I was stuck at a problem. Thanks are also due to PhD candidate Kristoffer Gryte for letting me use and giving me technical support for his implementation of the UAV model, and to PhD candidate Joo F. Fortuna for giving me an introduction to hyperspectral cameras.

And last but not least, I would like to thank my family for their support, even though they still, after five years, are not completely sure what cybernetics actually is.

Andreas Nordby Vibeto
Trondheim, June 5th, 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAV) are today widely used in ground observation, and by equipping them with different sensors they can be used in different situations. While the use of UAVs eases many cases of ground observation, there are some difficulties related to the attitude of the aircraft. When the sensor is attached directly to the aircraft the sensor will be coupled with the UAVs states, so what is captured by the camera depends on the angles of the UAV. Figure 1.1 illustrates how what is captured by the camera changes with the roll angle ϕ.

A common solution to decouple the sensor from the UAV states is to attach the sensor to a gimbal which will counteract the movements of the UAV. While this is a good solution for decoupling, it raises some new issues regarding its weight and size. As one of the benefits of UAVs is their small size, the gimbal can quickly be too big and heavy for the UAV, and it may make the aerodynamics of the aircraft less effective. This usually leads to increased fuel consumption.

This paper will investigate methods that will ensure precise ground observation when a camera attached directly to the aircraft is used to observe both curved and piecewise linear paths on ground level, while also avoiding the extra costs associated with a gimbal. This will be accomplished by finding an optimal path that minimizes the deviance between what is to be observed and what is captured by the camera. The optimal path will be calculated by an offline intervalwise Nonlinear Model Predictive Control (NMPC) algorithm before the flight commences [1]. The control method is developed with the usage of a hyperspectral pushbroom camera that is fixed to the UAV in mind.

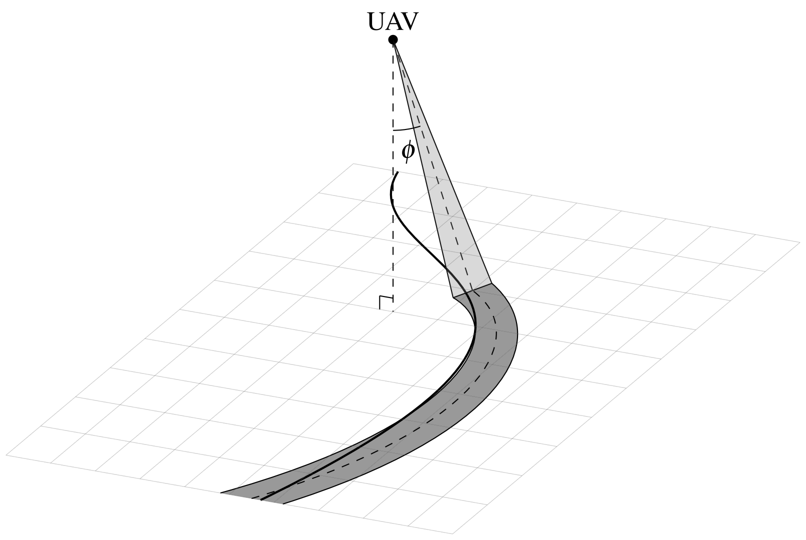


Figure 1.1: An illustration of the issues related to UAV operation with fixed sensors.

1.1 Related Work

The most common method to decouple the UAV attitude states from the sensor today is to equip the aircraft with a gimbal, which results in easy UAV operation without losing track of the features that is to be observed. Since the gimbal angles have limited range, features can be lost from the camera field of view (FOV) for some combinations of aircraft positions and attitudes. Previous solutions to this problem include generating trajectories that ensures that the gimbal angles are able to cover the features of interest [2], or putting constraints on the roll angle and altitude of the UAV [3].

A simpler solution to avoid lateral movements of the FOV is to change the UAV course by using the rudder instead of the ailerons. The rudder deflection creates a yawing moment which causes the aircraft to change course [4]. This type of controller is referred to both as a Rudder Augmented Trajectory Correction (RATC) controller [4] and a skid-to-turn (STT) controller [5]. Results show that the performance of these controllers are comparable to conventional controllers using roll to change course, and that errors in the images is greatly reduced [4] [5] [6].

While the controllers offer a solution to the control problem that reduces the errors in the images, they do not ensure that the features of interest stay inside the sensors FOV. To ensure that they stay within the FOV, Jackson has developed an optimization algorithm that minimizes the error of the sensor footpring [7]. Jakcson's solution differs from the solution proposed in this paper as his solution uses a simplified model of the UAV to calculate the optimal path online, while this paper focuses on using a more precise model to find an optimal solution before the flight is initiated.

Jackson presents a path planner that aims to minimize the error between the target on the ground, and the footprint of a camera fixed to a UAV by using a Nonlinear Model Predictive Controller (NMPC). The NMPC is compared to a PID and a sliding-mode controller that seek to follow the same path. Simulations of the three controllers show that of the three, the PID controller had the biggest crosstrack error. The results of the simulation of the NMPC and the sliding-mode controller showed that the two had comparable performance. The NMPC controller was able to find a near optimal solution with the performance characteristics of a real-time application.

One important point made by Jackson is that perfect tracking of a ground path with a fixed camera is not possible when using the roll angle to change the course of the aircraft. A controller that attempts to solve this problem would be unstable beacause of the depence the camera position has on the roll angle. When the path turns right and the aircraft rolls right to follow this path, the camera position will move left, away from the path. This only applies to controllers that attempt a perfect tracking of the path, so that a near-perfect tracking is still achievable.

1.2 Hyperspectral Imaging

The control method developed in this paper will be developed with the use of a fixed hyperspectral, pushbroom sensor in mind. A hyperspectral sensor/camera allows for accurate detection of different types of material from the UAV by sensing the wavelength of the received light.

1.2.1 Description

Hyperspectral imaging uses basics from spectroscopy to create images, which means that the basis for the images is the emitted or reflected light from materials [8]. The amount of light that is reflected by a material at different wavelengths is determined by several factors, and this makes it possible to distinguish different materials from each other. The reflected light is passed through a grate or a prism that splits the light into different wavelength bands, so that it can be measured by a spectrometer.

When using a hyperspectral camera for ground observation from a UAV, it is very likely that one pixel of the camera covers more than one type of material on the ground. This means that the observed wavelengths will be influenced by more than one type of material. This is called a composite or mixed spectrum [8], and the spectra of the different materials are combined additively. The combined spectra can be split into the different spectra that it is build up of by using noise removal and other statistical methods which will not be covered here.

1.2.2 UAV Ground Observation

Hyperspectral imaging is already being used for ground observation from UAVs. Its ability to distinguish materials based on spectral properties means that it can be used to retrieve information that normal cameras are not able to. For example in agriculture it can be used to map damage to trees caused by bark beetles [9], or it can be used to measure environmental properties, for example chlorophyl fluorescense, on leaf-level in a citrus orchard [10].

Systems for ground observation with hyperspectral cameras can be very complex, which often leads to heavy systems. In [11], a lightweight hyperspectral mapping system was created for the use with octocopters. The purpose of the system is to map agricultural areas using a spectrometer and a photogrammetric camera, and the final takeoff weight of the system is 2.0 kg. The resolution of the final images made it possible to gather information on a single-plant basis, and the georeferencing accuracy was off by only a few pixels.

1.3 Contributions

This paper will investigate how a UAV can best be controlled in order to ensure that what is capture by a fixed hyperspectral camera is what is intended to be captured. The goal of the thesis is to develop software able to generate an optimal path that can be used to survey a pre-define ground path. The contributions from the thesis are:

- A kinematic model to calculate the position of the camera footprint on the ground based on the attitude of the UAV
- An offline intervalwise MPC that, given a pre-defined discretized ground path, calculates an optimal path the UAV may follow in order to ensure the entire ground path is captured by a fixed camera
- A software implementation of the offline intervalwise MPC implemented using the ACADO Toolkit
- A Software-in-the-Loop simulator to test the performance of the path using an pre-existing autopilot

1.4 Thesis Outline

In chapter 2 the equations for the UAV model used in this thesis will be given, as well as the kinematic model used to compute the camera position on the ground. In chapter 3 the equations needed to create an offline intervalwise MPC that seeks to minimize the distance between the camera centre point on the ground and the ground path that is to be observed will be given. In chapter 4 the software implementation of the MPC using the ACADO Toolkit will be presented, and in chapter 5 the simulation environment will be presented.

In chapter 6 the performance of the software implementation of the MPC will be analyzed to state what it is capable of. Some of the paths optimized in chapter 6 will in chapter 7 be used to guide an UAV in simulations. Discussion and conclusion of the thesis will be given in chapter 8 and 9 respectively.

The complete state space model of the UAV will be presented in appendix A. In appendix B the optimization algorithm implemented using the ACADO Toolkit will be presented, and in appendix C the code used to create the offline intervalwise MPC will be presented.

Chapter 2

Kinematics

What is captured by the camera, the camera footprint, is dependent on the attitude angles of the aircraft when the camera is fixed to the aircraft body. In this section a model for calculating the camera footprint on the ground assuming flat earth will be presented, as well as the necessary UAV states for this thesis.

2.1 Reference Frames

Three different reference frames will be used to describe the kinematics of the UAV: the *body frame*, *North East Down* (NED) frame and *Earth Centered Earth Fixed* (ECEF) frame. The transformations given here can be found in Fossen [12].

The body frame, denoted $\{b\}$, is attached to the UAV and is used to describe the attitude and velocity of the aircraft. The NED frame, denoted $\{n\}$, is used to locally describe the position of the UAV using Cartesian coordinates. The position of the camera footprint will be given in the NED frame, based on the attitudes in the body frame.

While the body and NED frame are local frames that are useful to describe the UAVs attitude, speed and position, a different frame is needed to express the location of the UAV in a global perspective. For this the ECEF frame, denoted $\{e\}$, is used. In the ECEF frame position is often represented using Cartesian coordinates with the origin at the earth center, but in this case the position will be represented by longitude, latitude and height.

2.1.1 Transformations

Between NED and Body

The body frame and the NED frame are related by rotation matrices, one for each of the attitude angles. The transformation to the NED frame from the body frame is given by the rotation matrix $\mathbf{R}_b^n$:

$$\mathbf{R}_b^n(\boldsymbol{\Theta}_{nb}) = \begin{bmatrix} c_\psi c_\theta & -s_\psi c_\phi + c_\psi s_\theta s_\phi & s_\psi s_\phi + c_\psi c_\phi s_\theta \\ s_\psi c_\theta & c_\psi c_\phi + s_\phi s_\theta s_\psi & -c_\psi s_\phi + s_\theta s_\psi c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{2.1}$$

where c and s are the cosine and sine trigonometric functions of the angle in subscript, respectively. The transformation from the NED frame to the body frame can be found by taking the inverse of the transformation matrix $\mathbf{R}_b^n(\boldsymbol{\Theta}_{nb})$.

Between NED and ECEF

The transformation between ECEF and NED frames when the position is given using longitude, latitude and height is also given by a rotation matrix $\mathbf{R}_n^e(\boldsymbol{\Theta}_{en})$. However, it is the velocity vectors in each frame that are related by the rotation matrix. The rotation matrix is composed by two rotations about the latitude l and longitude μ:

$$\mathbf{R}_n^e(\boldsymbol{\Theta}_{en}) = \begin{bmatrix} -c(l)s(\mu) & -s(l) & -c(l)c(\mu) \\ -s(l)s(\mu) & c(l) & -s(l)c(\mu) \\ c(\mu) & 0 & -s(\mu) \end{bmatrix}. \tag{2.2}$$

The transformation between the velocity vectors in the ECEF and NED frame can be written as:

$$\dot{\mathbf{p}}_{b/e}^e = \mathbf{R}_n^e(\boldsymbol{\Theta}_{en})\dot{\mathbf{p}}_{b/e}^n. \tag{2.3}$$

Since this transformation represent velocities, a reference position must be known when transforming positions. Since NED is a local frame, the position in NED $\mathbf{p}^n$ will be given as a displacement from the reference ECEF position $\mathbf{p}_0^e$. The relation between position in NED and ECEF can therefore be written as [13]:

$$\mathbf{p}^e - \mathbf{p}_0^e = \mathbf{R}_n^e(\boldsymbol{\Theta}_{en})\mathbf{p}^n. \tag{2.4}$$

2.2 UAV Model

In this thesis the linearized UAV model presented by Beard & McLain [14] will be used as the prediction model for the path planner that is presented in chapter 3. In this section the UAV states used in this model will be presented, as well as the linearization method.

2.2.1 UAV States

The position of the UAV will be given using the North East Down (NED) coordinate frame denoted $\{n\}$:

$$\mathbf{p}^n_{b/n} = \begin{bmatrix} p_N \\ p_E \\ p_D \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix}, \tag{2.5}$$

while the global location will be given using the Earth Center Earth Fixed (ECEF) frame, denoted $\{e\}$, represented by longitude and latitude:

$$\mathbf{\Theta}_{en} = \begin{bmatrix} l \\ \mu \end{bmatrix}. \tag{2.6}$$

Following the notation used in [14], the velocities of the UAV will be given in the body frame denoted $\{b\}$:

$$\mathbf{V}^b_g = \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \tag{2.7}$$

The attitude $\mathbf{\Theta}_{nb}$ of the UAV will be given as Euler-angles, with the corresponding angular velocities $\dot{\mathbf{\Theta}}_{nb}$:

$$\mathbf{\Theta}_{nb} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \quad \dot{\mathbf{\Theta}}_{nb} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{2.8}$$

Euler angles are used over quaternions in this paper because the optimization is to be run offline so that computation time is not a critical measure. Even though Euler angles do suffer from gimbal lock while quaternions don't [14], the UAV will not be performing any high-angle maneuvers so that a gimbal lock should never occur.

The UAV model has four control inputs, namely the elevator, aileron, rudder, and throttle:

$$\mathbf{u} = \begin{bmatrix} \delta_e & \delta_a & \delta_r & \delta_t \end{bmatrix}^\intercal . \tag{2.9}$$

The complete nonlinear state space equations for the UAV model is given in appendix A.1.

2.2.2 Linearizing the Model

In order to ease computational load the *linear decoupled model* of a UAV, presented by Beard & McLain [14], will be used in this thesis. While a linear model is not able to fully describe the motions of an aircraft, it is valid around the *trimmed state* of the aircraft. An aircraft in its trimmed state will be able to maintain a straight level flight without any change in the control input, and since the UAV in this thesis is not expected to perform any high-angle maneuvers that puts it far away from the trimmed state, the linear model will be valid. An aircraft in the trimmed state satisfies the following equation:

$$\dot{\mathbf{x}} = f(\mathbf{x}^*, \mathbf{u}^*) = 0. \tag{2.10}$$

Linearization is performed by adding perturbations to the trimmed state solution, and the linearized states $\bar{\mathbf{x}}$ represent the perturbations away from the trimmed state [15]. The linearized state is defined as $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}^*$, where $\mathbf{x}^*$ is the trimmed state.

The states of an aircraft are highly *coupled*, meaning that they affect each other. This greatly increases the complexity of finding an optimal solution of the model since a change in one variable has effect on more than one state. For this reason the model is also decoupled into lateral and longitudinal models, where the states in one of the models does not affect the states in the other model. This simplification is done by removing terms that has a very small effect on the state, as these effects are easily controlled by the control systems [14]. The lateral and longitudinal states are given as:

$$\begin{aligned} \dot{\mathbf{x}}_{lat} &= \begin{bmatrix} v & p & r & \phi & \psi \end{bmatrix}^\intercal, \mathbf{u}_{lat} = \begin{bmatrix} \delta_a & \delta_r \end{bmatrix}^\intercal \\ \dot{\mathbf{x}}_{lon} &= \begin{bmatrix} u & w & q & \theta & h \end{bmatrix}^\intercal, \mathbf{u}_{lon} = \begin{bmatrix} \delta_e & \delta_t \end{bmatrix}^\intercal . \end{aligned} \tag{2.11}$$

2.3 Camera Footprint

The camera footprint is coupled with the three attitude angles given in $\mathbf{\Theta}_{nb}$. The position of the camera footprint will be calculated using forward kinematics, and an illustration of how the roll ϕ and pitch θ affects the camera position is shown in Figure 2.1.

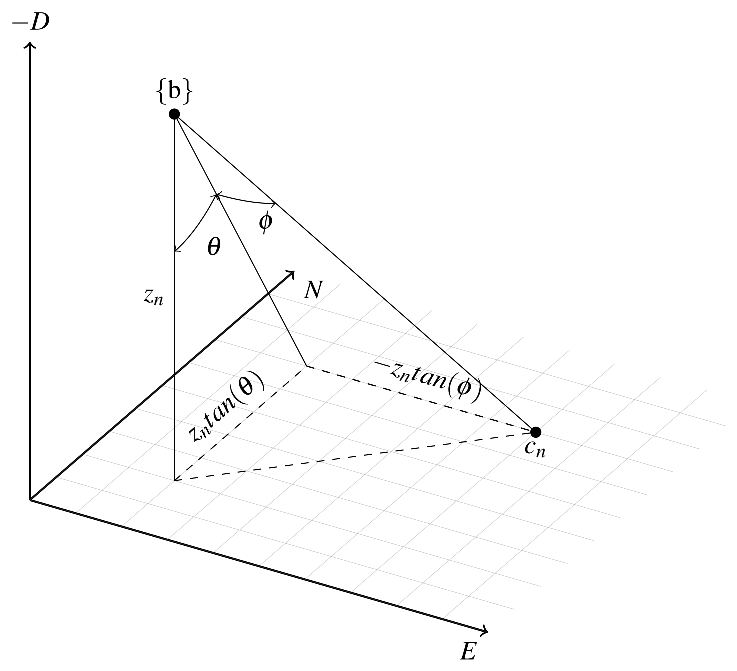


Figure 2.1: Illustration of how the aircraft attitude influence the camera position.

2.3.1 Centre Position

The attitude of the UAV is given in the body frame $\{b\}$ and the height z_n is given in the NED frame $\{n\}$, and the model presented here assumes flat earth. The position of the footprint centre point $\mathbf{c}_b^b$ in the body frame $\{b\}$ can be expressed as the distance from the center of the body frame, the UAV, caused by the angles ϕ and θ:

$$\mathbf{c}_b^b = \begin{bmatrix} c_{x/b}^b \\ c_{y/b}^b \end{bmatrix} = \begin{bmatrix} z_n tan(\theta) \\ -z_n tan(\phi) \end{bmatrix}. \tag{2.12}$$

The coordinates of the camera position in $\{n\}$ can be found by rotating the point $\mathbf{c}_b^b$ with respect to the aircraft heading ψ, and by translating the rotated point to the aircrafts position in the $\{n\}$ frame. The rotation matrix for rotating with respect to the heading is given as:

$$\mathbf{R}_{z,\psi} = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix}. \tag{2.13}$$

The final expression for the camera footprint centre position $\mathbf{c}^n$ in the $\{n\}$ frame then becomes:

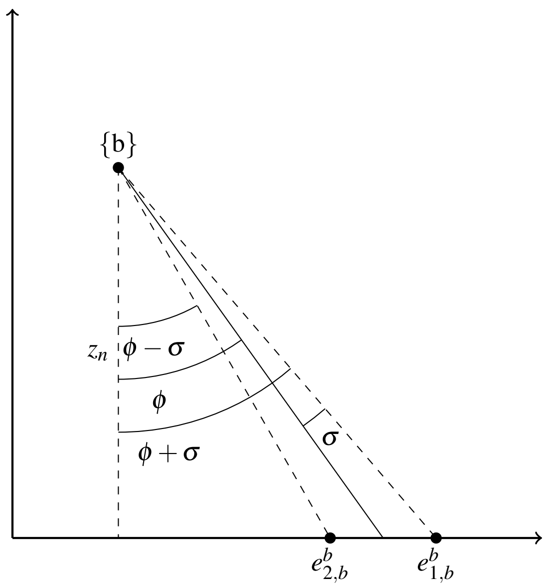


Figure 2.2: Illustration of how the field of view for a pushbroom sensor is calculated.

$$\begin{aligned} \mathbf{c}^n &= \mathbf{p}^n + \mathbf{R}_{z,\psi}\mathbf{c}^b_b \\ &= \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \mathbf{R}_{z,\psi} \begin{bmatrix} x^b_{x/b} \\ c^b_{y/b} \end{bmatrix} \end{aligned} \tag{2.14}$$

2.3.2 Edge Points

Since a hyperspectral pushbroom sensor captures images in a line, the centre point of the camera footprint does not express the entire area that is captured by the sensor. The edge points of the camera footprint are calculated with respect to the sensor's field of view, as shown in figure 2.2. These points, **e**, can be found by altering (2.12):

$$\mathbf{e}^b_{1,b} = \begin{bmatrix} z_n tan(\theta) \\ -z_n tan(\phi + \sigma) \end{bmatrix}, \ \mathbf{e}^b_{2,b} = \begin{bmatrix} z_n tan(\theta) \\ -z_n tan(\phi - \sigma) \end{bmatrix}. \tag{2.15}$$

The final expression for the cameras edge points then becomes:

$$\mathbf{e}^n_i = \mathbf{p} + \mathbf{R}_{z,\psi}\mathbf{e}^b_{i,b}. \tag{2.16}$$

Chapter 3

Model Predictive Control

Model Predictive Control (MPC) is a term used to describe control methods that uses knowledge about the process to calculate the future control inputs to the system in order to follow a reference trajectory [16]. In this chapter the equations for an *offline intervalwise MPC* that seeks to minimize the distance between the camera centre point and the ground path that is to be observed will be given. A linear state space-model for the UAV will be used to predict the future states and control inputs.

3.1 MPC Method

The MPC strategy can be broken down into three tasks [16]:

1. Predict the future outputs of the process for the given prediction horizon using past inputs to the process and the past measured states of the process, and by using the future control signals.
2. Optimize an objective function in order to determine the future control signals that follows a given reference trajectory as closely as possible.
3. Apply the optimal control signals to the process, and measure the resulting output so that it may be used to calculate the next prediction horizon in the first task.

In short MPC problems are made up of three elements: Prediction model, objective function and constraints. The prediction model represents the model of the process that is to be controlled, and will in this case consist of the differential equations for the states of the UAV. The objective function is the function that is to be minimized by the optimization algorithm, in this case this will be the distance from the camera centre point to the desired ground path together with some of the UAV states that will give a stable flight. The objective function represents the difference between the reference trajectory that the UAV is to follow and the current states of the UAV. The constraints are used to limit the values that either states or control inputs can take, and can prevent solutions that are not physically feasible.

A common mathematical formulation of the three elements that make up the optimization problem is shown in (3.1) [17]. $f(x)$ represents the objective function that is subject to equality and inequality constraints respectively. In this thesis the differential equations describing the UAV model will be implemented as equality constraints, while the inequality constraints will be used to define the ranges the control inputs must be within.

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{s.t} \quad & c_i(x) = 0, i \in \varepsilon, \\ & c_i(x) \geq 0, i \in I. \end{aligned} \tag{3.1}$$

3.2 Discretization

The model and optimization problem will be written on continuous time form, which means that it has to be discretized in order to be solved. The method used to discretize the problem plays a big role in how the problem is solved, and a common method to use for nonlinear programs (NLP) is the *direct multiple-shooting* method. Direct discretization methods can be explained as ”first discretize, then optimize”, which allows for easier treatment of inequality constraints [18]. One of the major benefits by using the direct multiple-shooting method is that all shooting nodes are initialized with the result from the previous iteration [19].

In short, the direct multiple-shooting method starts by computing a discretized control trajectory for a finite time interval. Independently, the Ordinary Differential Equations (ODE) of the optimization problem is solved one time for every timestep of the discretized control trajectory. Simultaneously, an integral of a cost function is computed, which is the reason why the direct multiple-shooting method is also called a *simultaneous* method.

$$\begin{aligned} \min_{\mathbf{s},\mathbf{q}} \quad & \textstyle\sum_{i=0}^{N-1} F_i(\mathbf{s}_i, \mathbf{q}_i) + E(\mathbf{s}_N) \\ \text{s.t} \quad & \mathbf{x}_0 - \mathbf{s}_0 = 0 \\ & \mathbf{x}_i(t_{i+1}; \mathbf{s}_i, \mathbf{q}_i) - \mathbf{s}_{i+1} = 0, \quad i = 0, \dots, N-1 \\ & h(\mathbf{s}_i, \mathbf{q}_i) \leq 0, \quad i = 0, \dots, N \end{aligned} \tag{3.2}$$

The direct multiple-shooting method can be described by the NLP shown in (3.2) [19]. In the equation the objective function F is the result of integrating the cost function, and **s** and **q** are the optimization variables for the states and controls respectively. **s** and **q** are introduced in order to ensure that the solution for the time interval is tied to the initial values. E is the end term of the objective function.

3.3 Offline Intervalwise MPC

The control problem in this thesis will be solved by using an offline intervalwise MPC to generate an optimal path that will reduce the image error when using a fixed camera to survey a ground track. The generated path is intended to be tracked by the autopilot on the UAV that will perform the survey, with the intention of optimally surveying the ground path.

3.3.1 Offline MPC

An *offline MPC* means that the initial state of the MPC is not a measurement of the UAV states, but rather the result of a simulation of the UAV. This means that the result from the prediction model used in the MPC will act as the physical system, and the outputs of the model will be fed back as inputs to the MPC for every iteration. The equations of the offline MPC are the same as the ones for the online version.

Rawlings & Mayne [20] refers to this kind of problem as a *deterministic problem* since there is no uncertainty in the system. A feedback loop in this kind of system is not needed in principle, since it does not present any new information. They also state that the resulting control action from an MPC for a deterministic system is the same as the control action from a *receding horizon control law* (RHC), which is another kind of predictive control.

3.3.2 Intervalwise MPC

Since this is a deterministic system, it is possible to perform the entire path optimization over one long optimization horizon. However, the computational load of using one long optimization horizon is heavier than the load of using several, shorter optimization horizons. For this reason an *intervalwise MPC* will be used. The term intervalwise has been introduced by Kwon & Han [1] to describe a type of receding horizon controller that implements the same strategy.

Commonly an MPC is used to optimize the model over a given *horizon*, where the initial states are given. After the optimization has finished, the first timestep of the optimization is returned and applied to the system, before a measurement of the system is performed. The new measurements are given as initial states for the next horizon, and so on.

The principle is the same for an intervalwise MPC. However, instead of only returning the first timestep, an *interval* of timesteps are returned, and the last timestep of the interval is used as intial states for the next optimization horizon. This way the number of MPC iterations is reduced, and the increased complexity by having long optimization horizons is avoided. Figure 3.1 shows how timesteps, intervals and horizons relate to each other. Since the MPC developed here is an offline MPC, the timesteps of each interval is stored as the result.

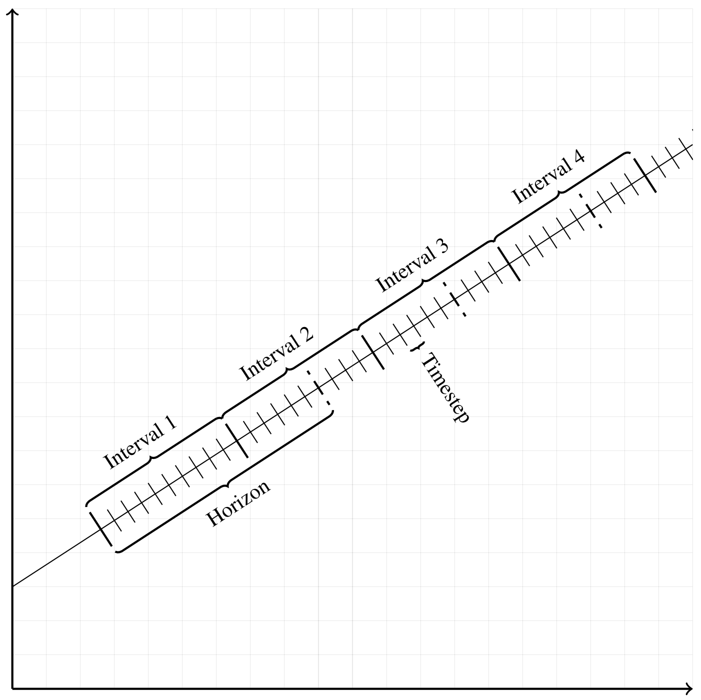


Figure 3.1: How intervals and horizons relate in an Intervalwise MPC.

3.4 Objective Function

The main objective of the MPC developed in this thesis is to minimize the cross track error between the centre point of the camera footprint and the ground path that is to be observed. This, together with other objectives, will be defined in the objective function of the optimization problem. In this section a way of formulating the objective function, least-squares, will be described, and how the objective function for an MPC is formulated to express the optimization horizons.

3.4.1 Least-Squares Problem

In many applications the objective function is formulated as a least-square (LSQ) problem. LSQ is a form of regression where the distance between a measurement and a known model is computed. In this case the known model is the reference signals, and the distance between the current states and the reference signal is calculated as a LSQ problem. The general mathematical formulation for LSQ is [17]:

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x) = \frac{1}{2}\sum_{j=1}^{m} |\phi(x,t_j) - y_j|. \tag{3.3}$$

In (3.3) r_j is called the residual function, which represents the distance between the measurement y_j taken at time t_j, and the model ϕ. In the optimization problem the residual function is what the algorithm seeks to minimize by selecting the parameters x that gives the lowest possible value of the residual function r_j.

In order to have a reference model that the measurements can be compared to, the desired values will be associated with timepoints. This means that the optimization algorithm will at given timepoints compare the current values of x to the value of the reference model at the same time. A visual representation of this is shown in Figure 3.2.

3.4.2 MPC Objective Function

The objective function is where the goal of the optimization is expressed, together with the optimization horizon of the problem. Typical goals of the optimization is to follow a predefined trajectory or reference signal while reducing the control inputs used. This can be expressed as follows [16]:

$$J(N_1,N_2,N_u) = \sum_{j=N_1}^{N_2} \delta(j)[\hat{y}(t+j|t) - w(t+j)]^2 + \sum_{j=1}^{N_u} \lambda(j)[\Delta u(t+j-1)]^2. \tag{3.4}$$

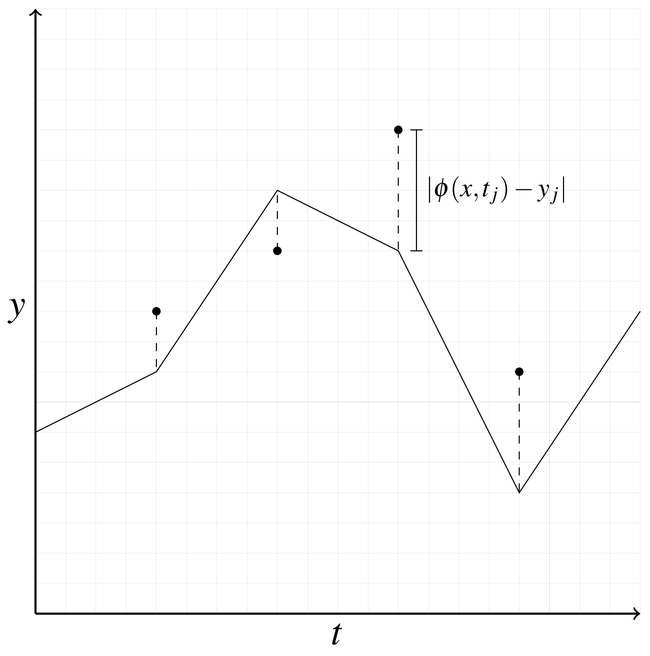


Figure 3.2: The distance between the measurements, represented by dots, and the model, represented by a line.

The first term of (3.4) represents the costs from the states of the model, and the second term represents the cost of the control effort. In the first term $\hat{y}$ is the value of the prediction model, which is compared to the desired trajectory w. In the second term the changes in control Δu is expressed. The change in control is used instead of the value of the control signal itself, since the steady state of the control signal may differ from zero. δ and λ are weighting variables which are used to tune the MPC. The three different N coefficients defines the horizon over which the states and the control effort should be optimized. The optimization horizon for states and control effort can be different, but they will stay the same for this problem.

3.5 Problem Definition

$$\begin{array}{rrcl}
\min\limits_{\mathbf{x},\Delta\mathbf{u}} & \multicolumn{3}{l}{\mathbf{J}_{i+k} = \frac{1}{2}\sum_{j=i}^{j+N}[(\mathbf{y}(\mathbf{x}_j)-\mathbf{y}_{d,j})^{\mathsf{T}}\mathbf{Q}(\mathbf{y}(\mathbf{x}_j)-\mathbf{y}_{d,j})+(\Delta\mathbf{u}_j)^{\mathsf{T}}\mathbf{R}(\Delta\mathbf{u}_j)]} \\
\text{s.t} & \mathbf{x}^{low} \leq & \mathbf{x}_j & \leq \mathbf{x}^{high} \\
& \mathbf{u}^{low} \leq & \mathbf{u}_j & \leq \mathbf{u}^{high} \\
& \Delta\mathbf{u}^{low} \leq & \Delta\mathbf{u}_j & \leq \Delta\mathbf{u}^{high} \\
& \dot{\mathbf{x}}_{j+1} & = & f(\mathbf{x}_j,\mathbf{u}_j)
\end{array} \tag{3.5}$$

The equations for the full optimization problem is shown in (3.5). The objective function uses the same setup as shown in (3.4), but in matrix form. Each of the three components of the problem definition will be described in detail in the following sections.

3.5.1 Prediction Model

The linear decoupled UAV model presented in chapter 2 will be used as the prediction model for the MPC. The model is associated with the following states and control inputs:

$$\mathbf{x} = \begin{bmatrix} p_N & p_E & h & u & v & w & \phi & \theta & \psi & p & q & r \end{bmatrix}^{\mathsf{T}} \tag{3.6a}$$

$$\mathbf{u} = \begin{bmatrix} \delta_e & \delta_a & \delta_r & \delta_t \end{bmatrix}^{\mathsf{T}}. \tag{3.6b}$$

The prediction model relates to the equality constraints of equation 3.1 in the form of differential equations. As explained in the previous chapter the control rates $\Delta\mathbf{u}$ will be used as control inputs in the optimization problem:

$$\Delta\mathbf{u} = \begin{bmatrix} \Delta\delta_e & \Delta\delta_a & \Delta\delta_r & \Delta\delta_t \end{bmatrix}^{\mathsf{T}}. \tag{3.7}$$

The control surfaces $\mathbf{u}$ are calculated from the rates $\Delta\mathbf{u}$ through integration:

$$\dot{\mathbf{u}} = \Delta\mathbf{u}. \tag{3.8}$$

3.5.2 Objective Function

The objective function $\mathbf{J}$ will be minimized over the entire optimization horizon, which consists of N timesteps. The current timestep for the entire optimization problem is denoted i, while the current timestep within the current optimization horizon is denoted j. Since this is an intervalwise MPC, as described in section 3.3, all states within in the interval will be stored, and the interval consists of k timesteps. If the number of intervals needed to cover the entire path is L, the result will contain kL timesteps.

The first term of the objective function calculates the distance between the UAV states and the reference trajectory. The vector $\mathbf{y}_d$ is the *measurement vector*, which is the references for the states:

$$\mathbf{y}_d = \begin{bmatrix} c_{xd} & c_{y_d} & h_d & u_d \end{bmatrix}^\intercal . \tag{3.9}$$

The function $\mathbf{y}(\mathbf{x})$ holds the current values for the optimization problem. While the height h and velocity u can be used as-is, the camera centre point $\mathbf{c}^n$ needs to be calculated using (2.14):

$$\mathbf{y}(\mathbf{x}) = \begin{bmatrix} p_N + h\cos(\psi)\tan(\theta) - h\sin(\psi)\tan(\phi) \\ p_E + h\sin(\psi)\tan(\theta) + h\cos(\psi)\tan(\phi) \\ h \\ u \end{bmatrix} . \tag{3.10}$$

In order to reduce the control effort for the optimization problem, the rate of change of the control inputs $\Delta\mathbf{u}$ will be minimized. Since all the control rates is to be compared to zero, no function is needed.

The matrices $\mathbf{Q}$ and $\mathbf{R}$ are the weighting matrices. They are diagonal matrices where each row represent one state or control rate. The higher the value in the row, the more value is given to the difference between the corresponding state or control rate and the reference trajectory while minimizing the objective function.

3.5.3 Constraints

The states and control inputs to the optimization problem are bounded by constraints to ensure that the values stays within ranges that are physically possible. The constraints $\mathbf{u}^{min}$ and $\mathbf{u}^{max}$ directly relates to the maximum deflection angle for the control surfaces, while the throttle is described as a proportion between zero and one. The same goes for the control rate constraints $\Delta\mathbf{u}^{max}$ and $\Delta\mathbf{u}^{min}$, as these as well are directly related to physical restrictions. It is worth noting that in addition to constraints, the control rates are included in the objective function, which seeks to minimize these variables.

When constraints are put on the optimization problem the complexity of the problem increases, which may make it more computational difficult to find a feasible solution.

For this reason the constraints put on the UAV states $\mathbf{x}^{min}$ and $\mathbf{x}^{max}$ will not be set to begin with, as it is assumed that the ”cheapest” way to fly the aircraft is the ”correct” way. However, if testing shows that the MPC finds solutions that shouldn’t be feasible, constraints will be included to remove these solutions.

Chapter 4

MPC Implementation

The offline intervalwise MPC presented in chapter 3 will be implemented using C++ and the ACADO Toolkit [21]. The implementation conists of two main parts: the MPC algorithm that prepares the optimization problem, and the optimization solver that will use the ACADO Toolkit to solve the optimization problem.

This chapter will describe the ACADO Toolkit, how to use it and how it works; as well as the MPC algorithm. An overview of what information the modules share is shown in figure 4.1.

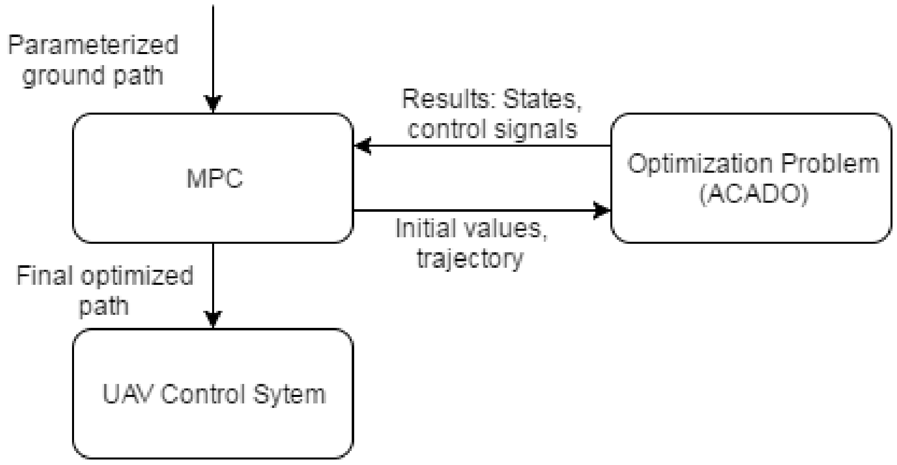

Parameterized ground path
Results: States, control signals
MPC
Optimization Problem (ACADO)
Initial values, trajectory
Final optimized path
UAV Control Sytem

Figure 4.1: An overview of what information the modules share.

4.1 ACADO toolkit

The ACADO Toolkit [21] is an open-source toolkit that supports several different methods for solving optimization problems. The toolkit provides methods to solve four different classes of optimization problem: Optimal control problems, multi-objective optimization and optimal control problems, parameter and state estimation problems, and model predictive control.

Even though the toolkit will be used to create an MPC in this paper, the optimal control problems (OCP) class will be used to solve the optimization problem. The reason for this is that between each iteration of the MPC algorithm a new trajectory must be generated, and the MPC problem class does not have the functionality needed to do this.

4.1.1 Runge-Kutta Method

The Runge-Kutta method is a form of *numerical integrator* that can be used to solve differential equations, and is used by the ACADO toolkit to integrate the prediction model. The method is based on the Euler method, which is a very simple method for numerical integration.

The ACADO toolkit provides algorithms for *explicit* Runge-Kutta methods [21], where explicit means that the method calculates the state of the system at a later time based on the current state. The Runge-Kutta method calculates the later state of the system by calculating several approximations of the derivative of the system. The current state of the system, together with a linear combination of the approximated derivatives, gives the next state of the system [15]. For a system on the form $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t)$, the Runge-Kutta method can be mathematically expressed as:

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}_n + h\sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j, t_n + c_i h), i = 1, ..., \sigma \tag{4.1a}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\sum_{j=1}^{\sigma} b_j\mathbf{k}_j \tag{4.1b}$$

where t_n is the current time, $\mathbf{y}_n$ is the output of the system at the current time, h is the step size and σ is the number of approximations of the derivative that is calculated. a, b and c are parameters for the specific Runge-Kutta method used, and c must satisfy $0 < c < 1$.

Systems may consist of dynamics of different time constants, and the difference between the constants may have a big impact on how fast an explicit method can perform the computations. While they don't have a formal definition, systems that explicit methods compute poorly are referred to as *stiff systems* [22]. The poor performance of explicit method comes from the stability of these methods, as the stability is dependent on the step size. This means that for systems with dynamics of varying time constants, a very small stepsize has to be chosen in order for the method to remain stable. For stiff systems *implicit* methods have a better performance and faster computation time.

4.2 Implementing the Optimization Problem

Since ACADO offers a symbolic way of implementing the optimization problem the equations from chapter 3 can be implemented as they stand. The 12 UAV states $\mathbf{x}$ are implemented as differential states, as well as the four control surfaces $\mathbf{u}$. The control rate $\dot{\mathbf{u}}$ is defined as the control states of the problem, and are linked to the control states through (3.8). The model of the UAV is implemented using the differential equations for each state.

ACADO offers a symbolic way of defining the objective function as an LSQ as well. By defining what states are included in the LSQ and their weights, ACADO automatically minimizes the objective function. The reference model can either be a constant value or a time varying variable. When a path is to be tracked, the path needs to be given with related timepoints.

Lastly, the constraints are also implemented using the symbolic syntax, by simply assigning max-min values for the variables that are subject to constraints. The initial value of the states is also set using the same syntax.

4.2.1 Nonlinear Prediction Model

Initially, effort was made to implement the nonlinear model presented by Beard & McLain [14] as the prediction model in the optimization problem. This would have given more precise results as the nonlinear model is a closer representation of the real UAV. Since the nonlinear model also includes the effect wind has on the aircraft, the path could be optimized with the knowledge about the wind conditions as well. The level of calculation needed for the nonlinear model is significantly higher; however, since this implementation is intended to run offline before the flight occurs, computation time is not a critical concern.

Achieving stable flight within the optimization problem with the nonlinear model on the other hand, turned out to be a difficult task that was far from trivial. This is somewhat due to the nonlinearity, but also to the high coupling between states in the model. The coupling causes changes in one state to affect many other states, which results in a much more complex problem. Several different algorithm and solver settings in ACADO was tested, as well as different objective functions and weighting of these functions. After many attempts the decision to use the linear model instead was made, largely due to this being a project with limited time available.

4.2.2 Nonlinearity

The ACADO toolkit is written for nonlinear optimization problems, and using it together with a linear optimization will give a correct solution, but the computation time for a linear problem will be longer than it needs to be because of extra overhead related to nonlinear algorithms [Cite sourceforge?].

For this reason, using it with a linear prediction model may seem odd. However, the cost function used in this problem is not linear. This is because of both the calculations for the position $\mathbf{p}_N$ and $\mathbf{p}_E$ is represented by nonlinear equations, and the equations to calculate the camera footprint are nonlinear. If the timing demands for this optimization problem was more important, a solution may be to linearize the position equations as well as the equations used to calculate the camera footprint, and then implement the optimization problem using a toolkit that is made for linear problems.

4.3 MPC

The task of the MPC module is to supply the ACADO implementation with the information needed to perform the optimization, and also control the optimization algorithm so that the correct horizon is calculated, as well as storing the results in the correct order. The pseudocode for the MPC implementation is shown in algorithm 1 and 2 in appendix C.

4.3.1 Generating the Trajectory

The ground path that is to be observed is assumed to be *time independent*, meaning that it does not matter when a section of the path is captured by the camera. However, the function that minimizes a least-squares objective function that is provided with the ACADO toolkit requires that the path is given as values with associated time points.

In order to meet this requirement a time dependent path will be generated at the beginning of every iteration of the MPC. This will be done by making the assumption that the UAV will maintain its reference speed throughput the horizon. With this assumption the distance the UAV will travel between every timestep can be calculated, and based on this distance the desired position for the UAV at the next timestep can be found. Since the horizon is in the order of seconds and the predicted path is updated every iteration, this assumption will not lead to big errors. The principle behind the calculation is shown in Figure 4.2.

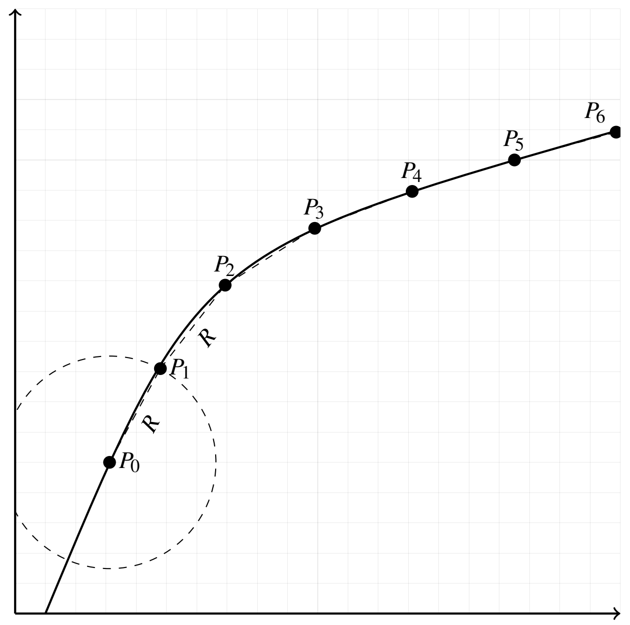


Figure 4.2: Calculating trajectory based on constant speed.

Chapter 5

Simulation Environment

5.1 Software In the Loop Testing

To test if the optimized path gives an improvement in ground observation, and if it is able to keep the observation path within the camera footprint at all times, software in the loop (SITL) testing will be performed. For this the applications Dune, ArduPilot and Neptus will be used. How they interact is shown in figure 5.1, and a short explanation of the three will be given in the following sections.

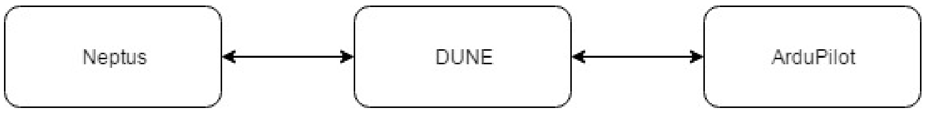


Figure 5.1: An overview of what information the modules share.

5.1.1 DUNE

DUNE Unified Navigation Environment is a part of the LSTS toolchain (Laboratrio de Sistemas e Tecnologia Subaqutica) which aims to provide a control architecture that will ease the control of unmanned air, ground surface and underwater vehicles [23]. DUNE is the software intended to run on board the unmanned vehicles, and provides sensor drivers, and navigation and control functionality. In this thesis DUNE will be used to turn the flight path into waypoints that can be sent to the autopilot.

The functionality needed to track the path is made as a *task* using the API provided by DUNE. The task receives information about the UAV states from the vehicle, and uses this information to provide meaningful input to the autopilot depending on the stage of the operation. When generating waypoints from the path the task uses the principle of *Line-of-Sight* (LOS) guidance to find waypoints a given distance away from the vehicle.

5.1.2 ArduPilot

ArduPilot is an open-source autopilot, which supports several types of vehicles [24]. It provides functionality for SITL simulation by interfacing the flight dynamics model provided by JSBSim [25]. ArduPilot communicates with DUNE to receive control commands and send information about the UAV states.

5.1.3 Neptus

Neptus is also a part of the LSTS toolchain, and is used to execute the simulations, and generate logs after they are finished [26]. It provides a map interface to observe the simulations in real-time, and also displays information about the aircraft. It communicates with DUNE throught using IMC messages based on a control message set defined by the LSTS toolchain.

5.2 Finding Trim Conditions

The trim conditions, as described in section 2.2, play an important role when using a linear model as this model is linearized about these points. Since the trim conditions also represent a straight level flight they are good initial states and controls for the simulation and are feasible points optimization. In order to find the trim conditions a built-in Matlab function together wtih a Simulink model will be used and the procedure that is used, which will be summarized in this section, is described by Beard & McLain [14].

Matlab features a built-in function `trim` that calculates the trim conditions of a given Simulink model. The Simulink model must be set up with the four control signals as inputs, and the output of the model is the airspeed V_a, the angle of attack α and the sideslip β. The out states are chosen as they easily expresses a trimmed stable flight. The airspeed is set to the desired cruise speed, while the angle of attack and sideslip is set to zero for a straight level flight. The function uses initial guesses of the states and inputs, also the derivatives, and what the desired output is. The Simulink model used for this thesis was developed by Gryte for his master thesis [27].

5.3 Generating the Path

The MPC application expects a parameterized path that can be either linear or curved. The linear paths will be generated using parameterization, and the curved paths will be generated as Dubins paths. This will be done using Matlab, and the generated paths will be stored in a textfile containing only the points on the path.

One important matter when generating the paths is to ensure that the resolution of the path is high enough. How high the resolution needs to be depends on the speed of aircraft and the length of the timestep, and how little uncertainty is needed when generating the trajectory.

Dubins Path

A Dubins path is a path that consists of two circles connected by a straight line, and it has been proven that this is the shortest path between two vehicle configurations [28]. In order to generate the Dubins path the algorithms presented by Beard & McLain [14] will be used. A n illustration of a simple Dubins path is shown in figure 5.2.

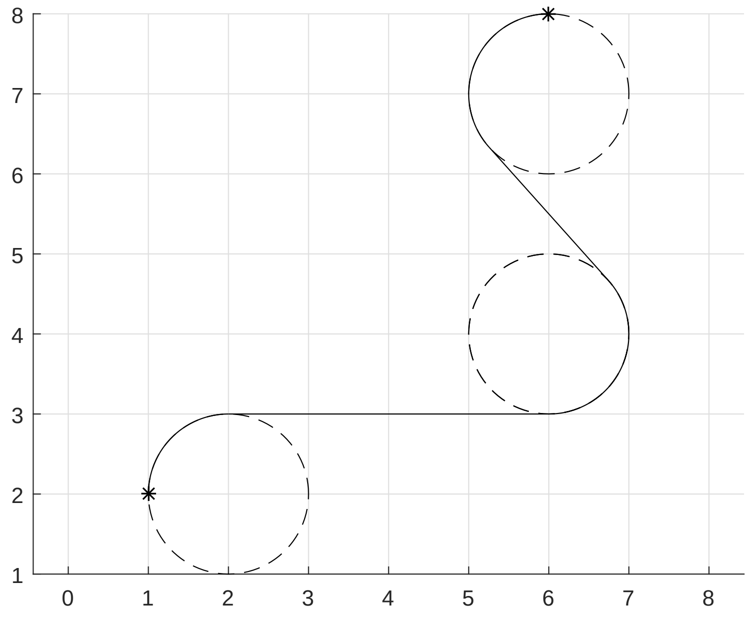


Figure 5.2: An illustration of a simple Dubins path.

Chapter 6

Optimizing Paths

In this chapter an analysis of paths optimized by the MPC will be performed. The MPC will optimize several paths, and the discussion will focus on how the control problem is solved.

The optimizations will be performed at an altitude of 150m unless stating otherwise. There are no explicit reason for choosing this altitude. For other survey missions using a hyperspectral camera the altitude has been as low as 100m [11], and as high as 1900m [29]. The cruising speed is set to 25m/s, as this gives a stable trimmed flight.

6.1 Horizon Length

In order to determine what horizon length is needed to optimize the path, several paths were optimized with horizon lengths varying from 10 to 140, spaced by 10. A piece-wise linear and a curved 45° turn was optimized with the different horizon lengths. The resulting path flown by the UAV is shown in Figure 6.1, and the resulting camera position can be seen in Figure 6.2.

The results for the two paths are very similar, and longer horizon length seems to be synonymous with better tracking of the ground path. The MPC compensates for the sideways shift in the camera position cause by the roll angle by initiating the turn earlier, and the aircraft levels out from the turn later for the same reason.

For the shorter horizon lengths the MPC runs into problems because it hasn't planned far enough ahead when the turn begins. As the path starts turning the aircraft is still straight above it. At this point it is too late to start altering the aircrafts position to ensure that the camera footprint stays on the path, so the MPC creates a roll angle in the opposite direction of the path so that the camera continues to move in the direction of the path. This in turn leads to the aircraft turning left, worsening the situation and making the problem more and more difficult. In some cases this causes the roll angle to become very high, causing the MPC to lose control and the aircraft loses height.

Upon closer inspections it can be seen that when the horizon length reaches 90, there are no more big unwanted motions in the system. As the horizon length reaches 110 the optimized paths are almost identical, and as can be seen in Figure 6.3b, there is very little precision to gain from increasing the horizon length. The optimization is time consuming, and as seen in Figure 6.3a the time it takes to optimize the paths increases exponentially. For these reasons a horizon length of 110 will be used for the rest of the simulations, as this gives an accurate path tracking for a relatively short computation time.

One result to take specific note of is the light blue UAV path in Figure 6.1a. This path is generated when the horizon length is 130, and is clearly not a solution to the optimization problem. There are no simple explanations for why this exact horizon length returns a bad result, as both lengths shorter and longer return good results. One explanation may be that this exact horizon length with this exact discretized path gives to a very unfortunate set of waypoints. This result has been removed from the camera position shown in Figure 6.2a, as it made it difficult to see the other correct results.

6.2 Turns

In this section both linear and curved turns of different degrees will be optimized, and an analysis will be performed on the results.

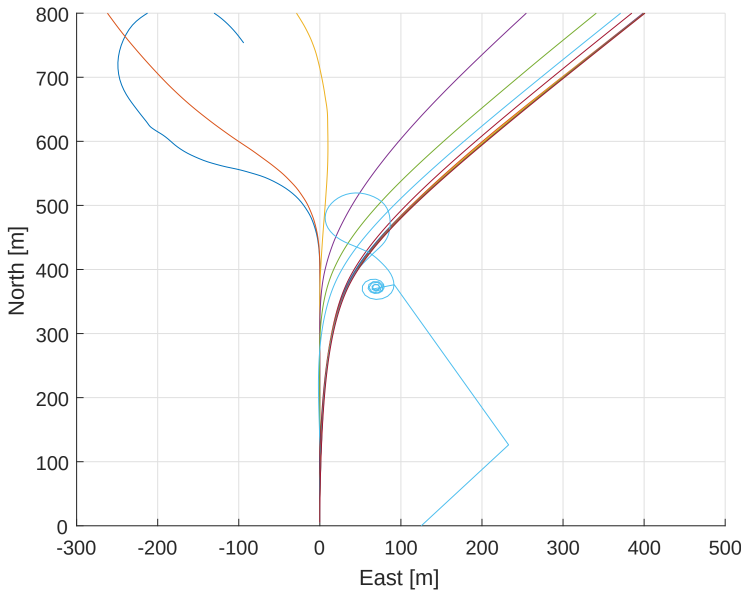
6.2.1 70° Turn

Curved 70° Turn

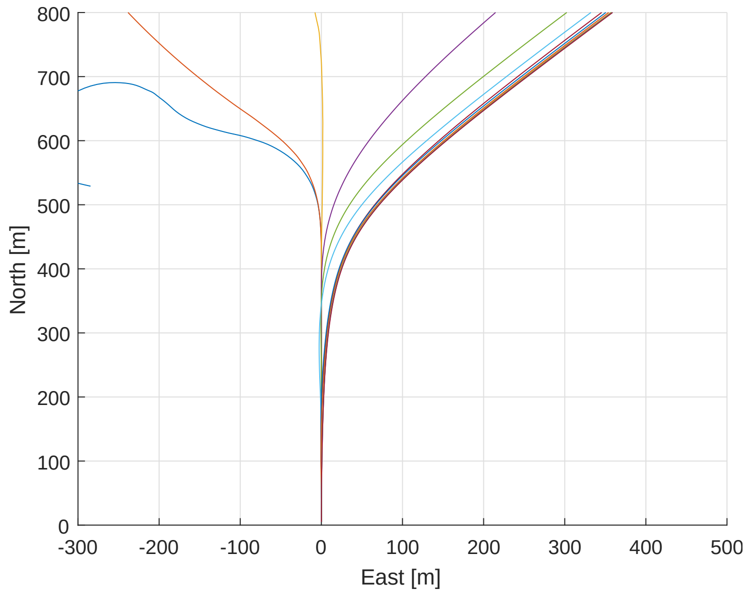
The optimized path of a 70° turn with a radius of 150m is shown in Figure 6.4. Even though the camera centre point deviates more from the desired path than during the 45° turn, the result is still a smooth path with no big unwanted motions. The aircraft maintains a stable altitude, and the angles are stable and smooth.

Linear 70° Turn

Optimizing a path containing a 70° linear turn returns a very different result than for the curved turn. As can be seen in Figure 6.5, the MPC is not able to achieve stable flight throughout the turn, and it is difficult to point to one exact reason for why the optimization fails. The aircraft is about halfway through the turn when it suddenly banks to the left and enters a spiral. This could be for the same reason as when the horizon length is too short, that it tries to cover the path by moving the camera position using roll. It is also possible that the optimization algorithm could not find a feasible

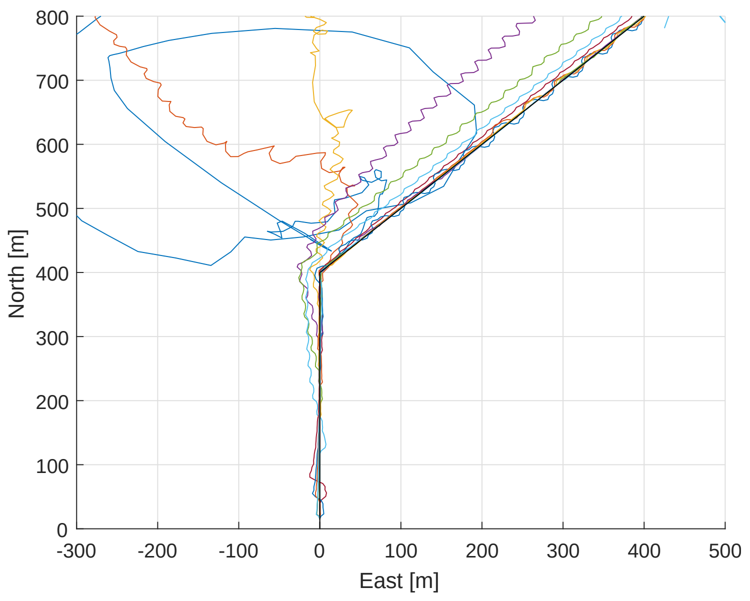


(a) Linear 45° turn.

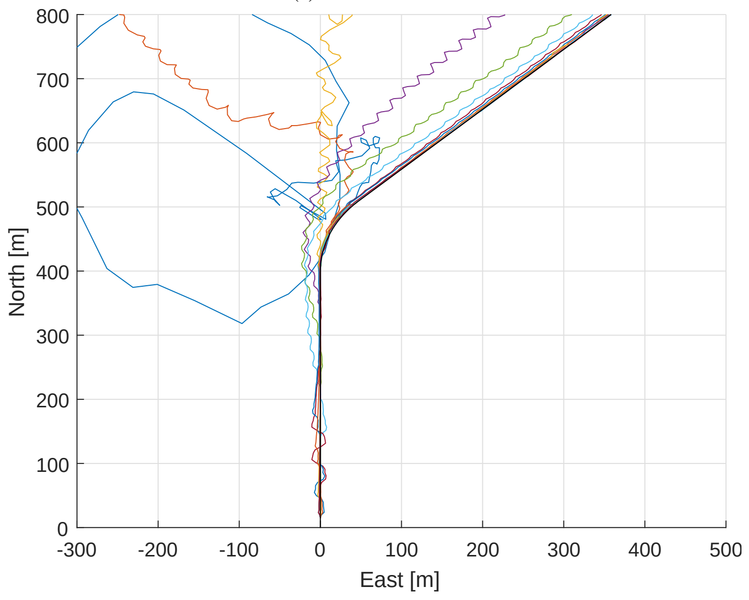


(b) Curved 45° turn with 200m radius.

Figure 6.1: The position of the UAV during the two turns with horizon lengths varying from 10 to 140.

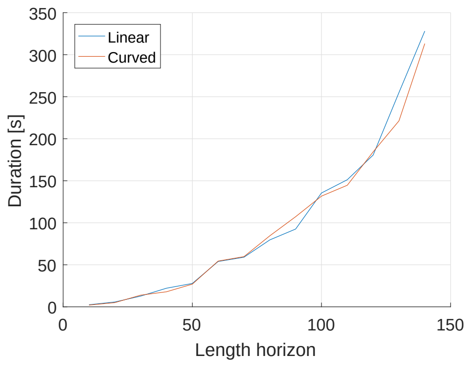


(a) Linear 45° turn.

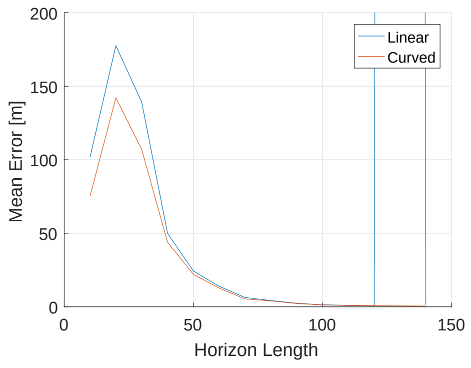


(b) Curved 45° turn with 200m radius.

Figure 6.2: The camera position during the two turns with horizon lengths varying from 10 to 140.

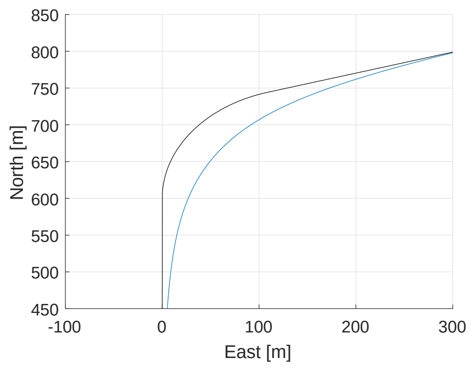


(a) Duration

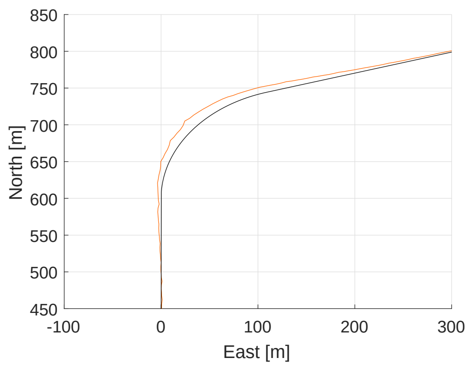


(b) Mean error

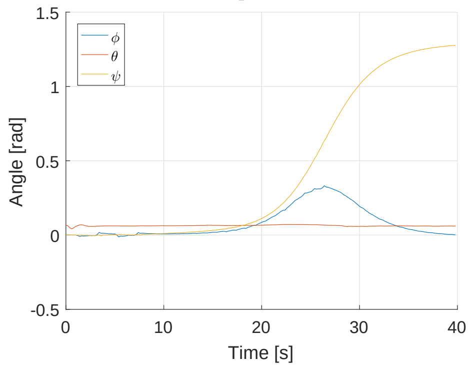
Figure 6.3: Duration of interval for varying horizon lengths, and the mean error.



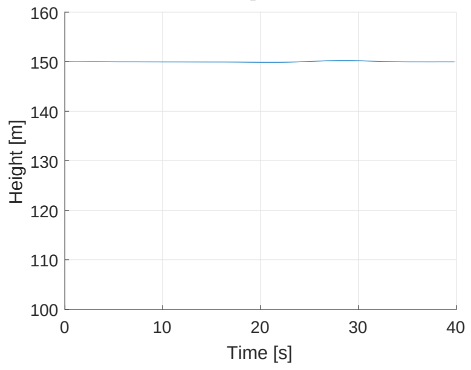
(a) UAV position.



(b) Camera poistion.

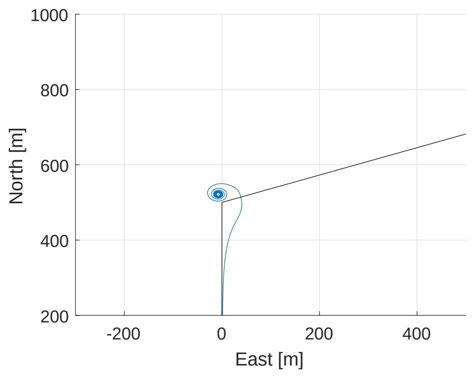


(c) Attitude angles

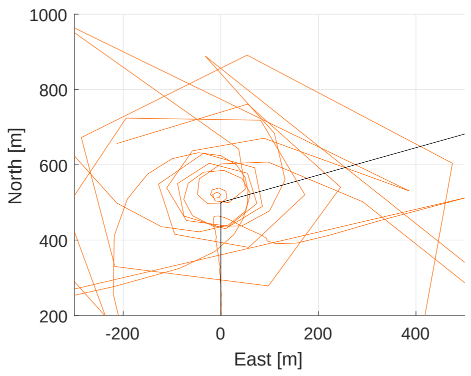


(d) Altitude

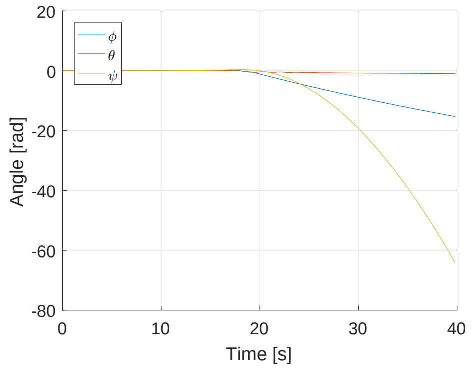
Figure 6.4: Results of optimizing a curved 70° turn.



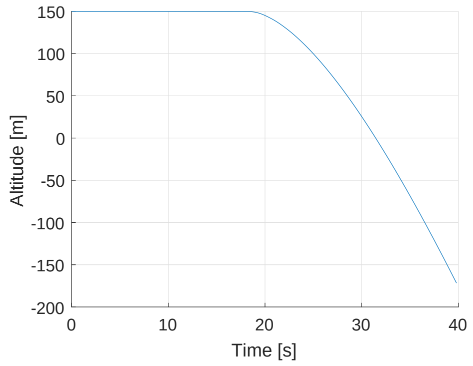
(a) UAV position.



(b) Camera poistion.



(c) Attitude angles



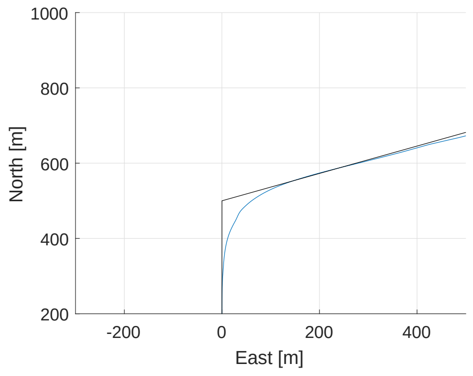
(d) Altitude

Figure 6.5: Results of optimizing a linear 70° turn with 10^{-1} weight on camera position.

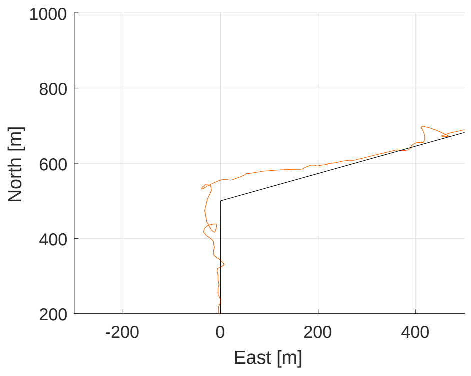
solution. When it cannot find a feasible solution it still returns some values and tries to continue, but in most cases the system is already unstable at this point, so it is not able to get it back to a stable state.

In an attempt to achieve stable optimization of the corner, the weighting of the camera position in the objective function was reduced. The result of changing the weighting from 10^{-1} to 10^{-3} can be seen in Figure 6.6. This tuning results in a stable flight, but the path tracking is not as precise and smooth as for the 45° turn. In addition the resulting camera path consists of several loops. This occurs as a combination of both the roll angle and pitch angle changing at the same time.

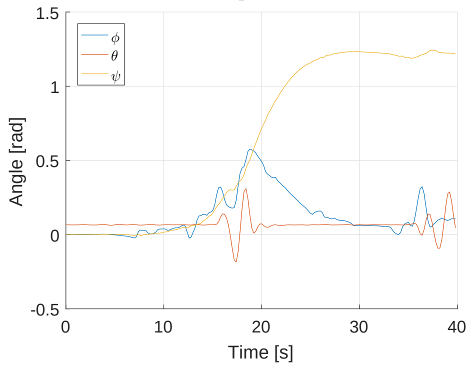
A third attempt on a linear 70° turn was made, this time with a weighting of the camera position of 10^{-5}. As can be seen in Figure 6.7 this results in a stable flight. The path tracking on the other hand, is poor. With this tuning the weight on the camera position is so much lower than on the other states included in the objective function, so that the path with the lowest cost is not the path that tracks the ground path.



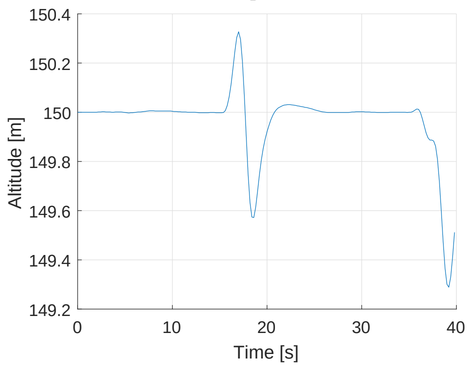
(a) UAV position.



(b) Camera poistion.

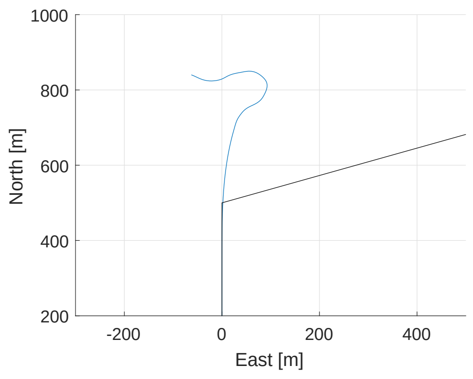


(c) Attitude angles

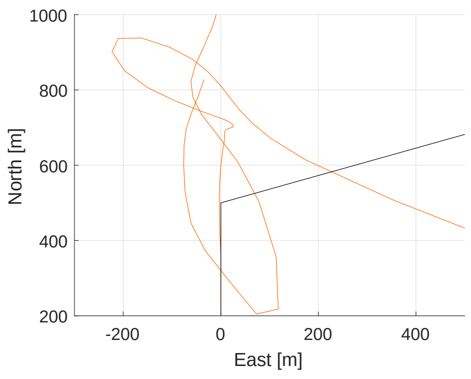


(d) Altitude

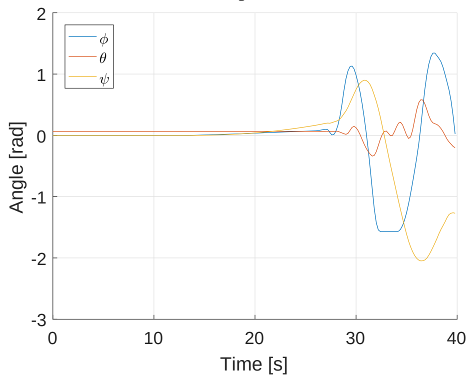
Figure 6.6: Results of optimizing a linear 70° turn with 10^{-3} weight on camera position.



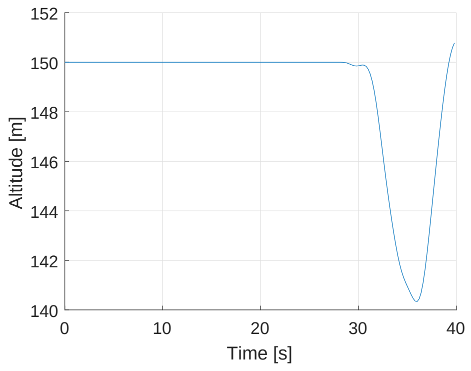
(a) UAV position.



(b) Camera poistion.



(c) Attitude angles



(d) Altitude

Figure 6.7: Results of optimizing a linear 70° turn with 10^{-5} weight on camera position.

6.2.2 90° Turn

Curved 90° Turn

Figures 6.8 and 6.9 shows the optimized paths of a curved 90° turn with varying radii. The results show that with a radius of 200m, 150m, and 50m the optimization algorithm returns a smooth stable flight path. As expected the camera position deviates more from the ground path than for gentler turns, and Figure 6.10 shows that the roll angles contains some smaller spikes that did not appear in the 70° turn.

For all the three radii there is a deviation between the camera centre point and the ground path after the turn. For the 50m radius turn in Figure 6.9d, this deviation is as high as 50m. While this most likely would correct itself of the path was simulated further, the deviation is a result of how the cost function is weighted. As mentioned in the previous section the weight on the camera position is so low compared to the weight put on the control rates, so having a deviation is cheaper than using control input to correct it.

A more surprising result is seen in Figures 6.8c and 6.9c. Even though the MPC is able to optimize the turn with 50m radius, it is not able to return a stable flight path for the turn with 100m radius. As for the linear 70° corner, the MPC appears to fail mid-turn, which causes the aircraft to enter a spiral motion. As the MPC handles the other radii well, this may occur because of a very unfortunate set of waypoints, as mentioned for the failing horizon length.

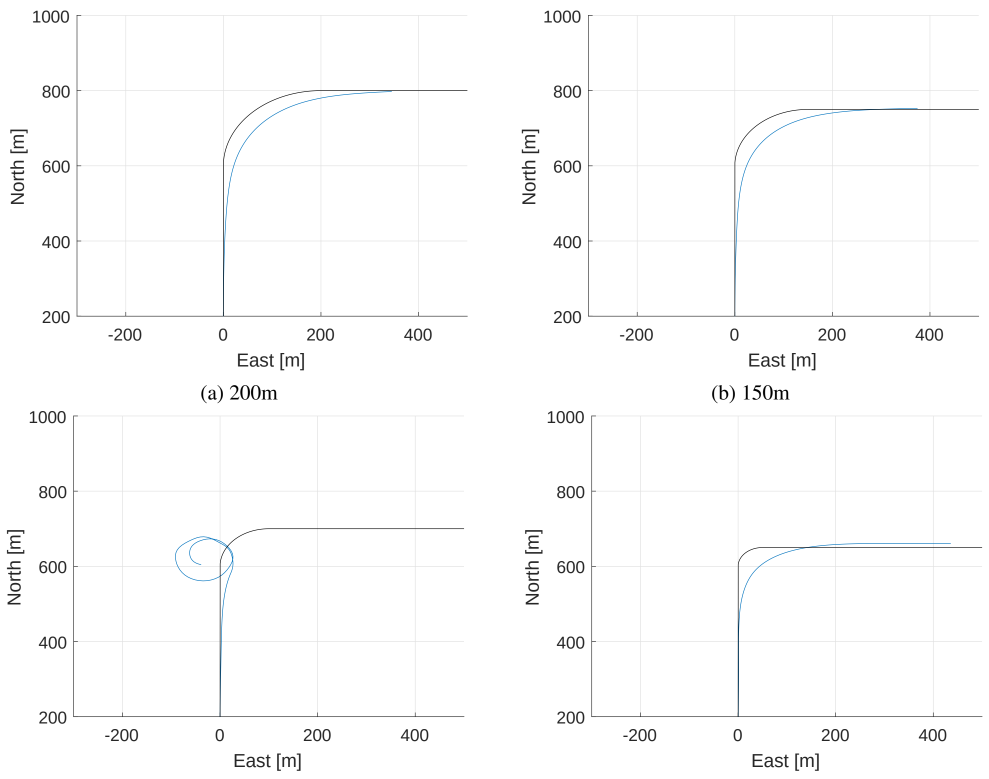
Linear 90° Turn

Since the MPC struggles to track a 70° corner, it is no surprise that it fails to track a 90° corner as seen in Figure 6.11. Once again the MPC ends up with solving a difficult path using roll instead of the position, which ends with a bad solution. Different weightings on the position was tested without success.

6.2.3 180° Turn

When optimizing a 180° turn the radius plays a big role, as can be seen in Figures 6.12 and 6.13. The figures show optimizations of six 180° turns with radius ranging from 300m to 50m.

Unsurprisingly, broader turns give better solutions. For the turn with 300m radius the tracking is very good. And while the tracking is not as good for the turns with 250m and 200m, they return a smooth stable path that puts the camera centre not too far away from the path. As for the 90° turn, the MPC accepts deviance after the turn rather than using expensive control inputs to correct it. Figure 6.14 shows that shorter radii require higher roll angles, which is as expected.



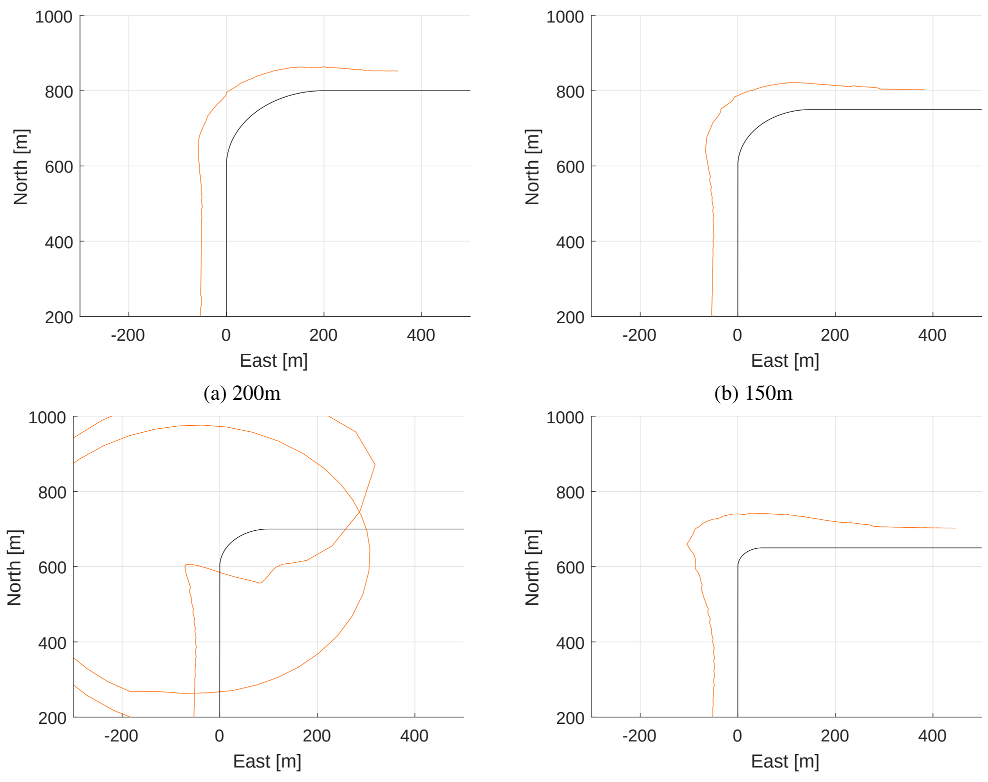
(a) 200m

(b) 150m

(c) 100m

(d) 50m

Figure 6.8: The position of the UAV when optimizing a curved 90° turn with varying radius.



(a) 200m (b) 150m (c) 100m (d) 50m

Figure 6.9: The position of the camera when optimizing a curved 90° turn with varying radius.

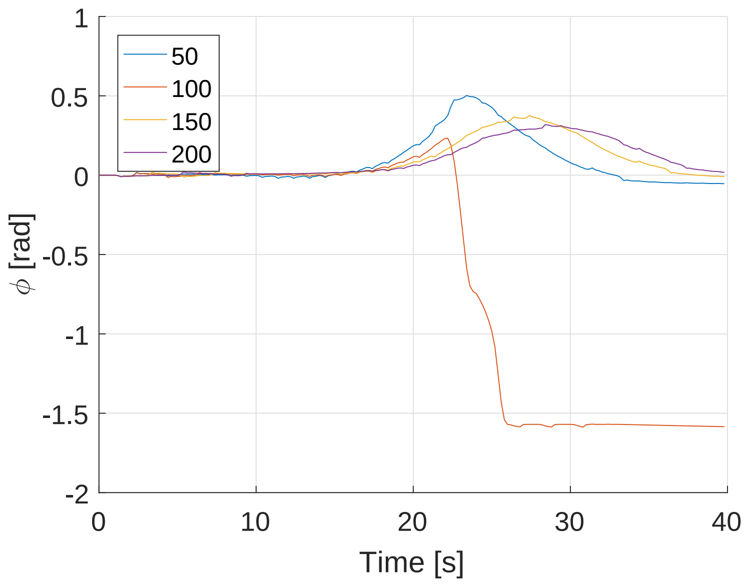
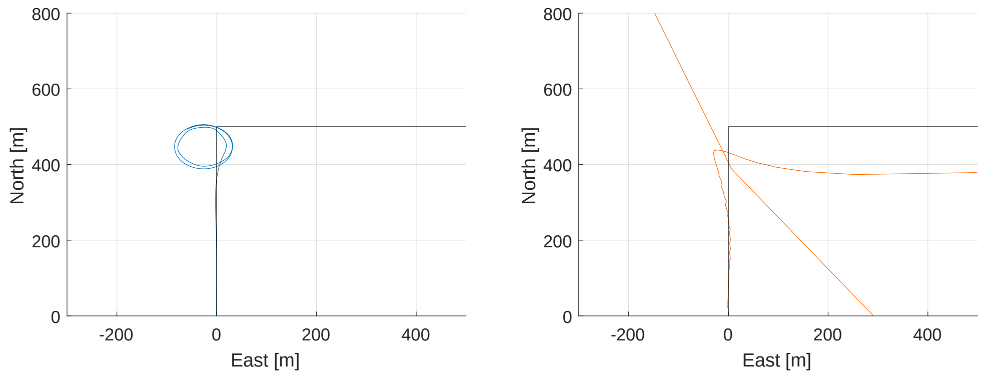


Figure 6.10: The roll angle ϕ during the 90° turns.



(a) UAV position.

(b) Camera poistion.

Figure 6.11: Result of attempting to optimize a linear 90° turn.

For the three shortest radii, 150m, 100m and 50m, the result is not as good. For the turns with 150m and 100m radius the result is similar to previous paths where the path is too sharp: the MPC fails halfway through and ends up spiralling in the opposite direction. In Figure 6.12e it may appear as the aircraft is about to recover and continue tracking the ground path, but closer inspection of the height plots show that at this point the aircraft is only about 20m above ground, and still descending.

For the 50m turn in Figure 6.12f the MPC fails before the turn has started. About 250m before the turn begins the aircraft drifts off to the left, the opposite direction of the turn. In Figure 6.13f it can be seen that the camera point is still close to the ground path at this point, however, it is swinging rapidly from side to side a few times before it completely drifts off.

6.3 Effect of Height

As can be seen in (2.12), the effect the pitch and roll has on the camera position increases proportionally with the altitude of the aircraft. In Figure 6.15, a 45° curved turn has been optimized with the aircraft flying at different altitudes, namely 100m, 200m and 300m. While the altitude do not seem to affect the path the MPC chooses to fly the aircraft, the effect is clearly visible in the camera position shown in Figure 6.15b. When flying at an altitude of 100m the camera position takes the inner turn, while for 300m it greatly widens the turn. For an altitude of 400m the MPC fails to return a stable result, which indicates that the optimization problem becomes more difficult with increasing altitude.

6.4 Path

In this section paths consisting of more than one turn will be optimized.

6.4.1 Two Opposite Turns

How the MPC takes advantage of subsequent opposite turns can be seen in Figure 6.16. When there is an opposite turn trailing the first turn, the optimized path cuts across the line connecting the two turns, while still keeping the camera on the observation path. This is also seen in Figure 6.16f that shows the course angle throughout the turn. The course angle never reaches 45° or 70°, which is the angle of the lines, because the MPC decides to cut across the lines and correct the camera position using roll angle. The course angle and heading angle are close to identical, which indicates that the MPC do not correct the course angle by using rudder to induce sideslip.

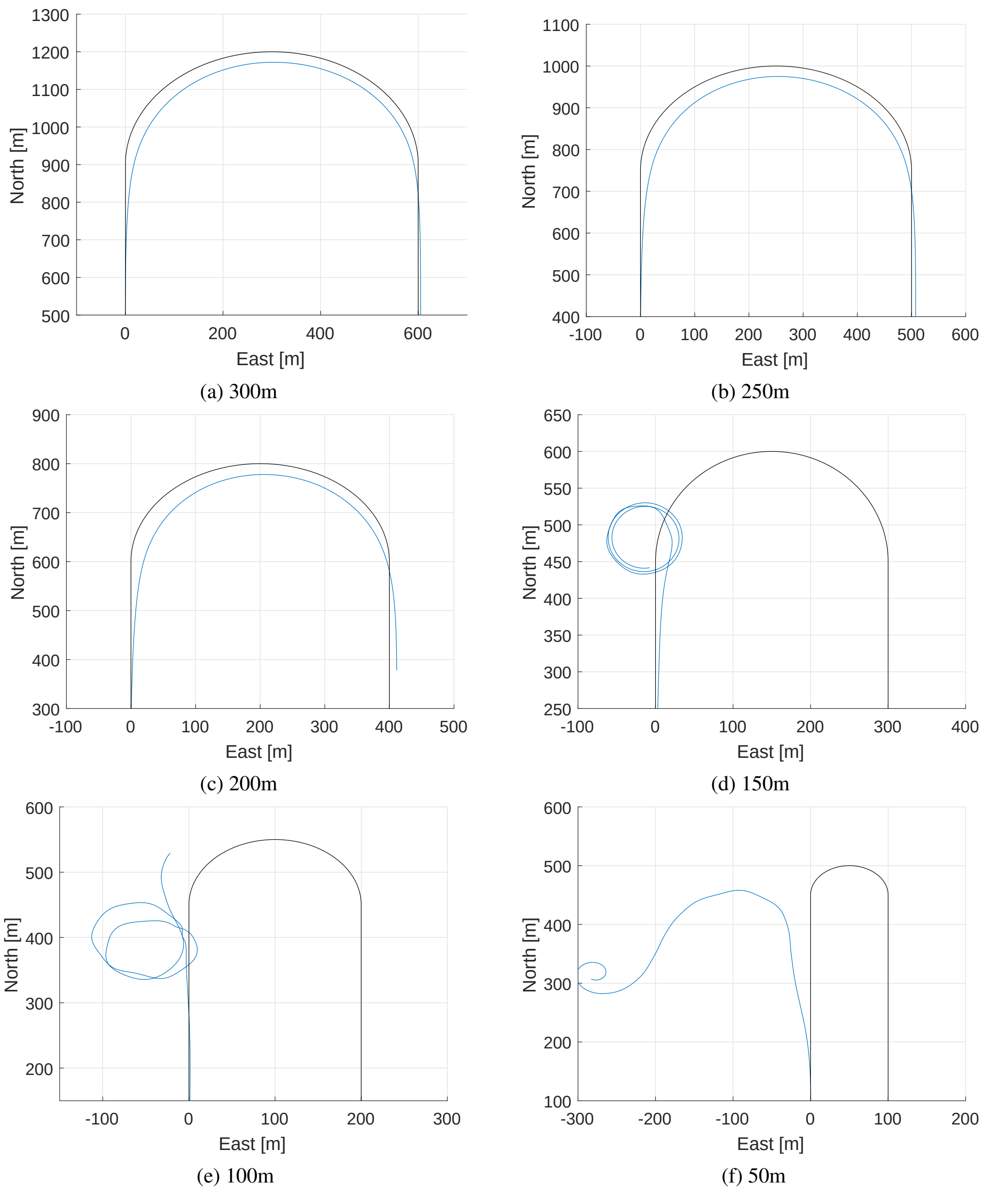
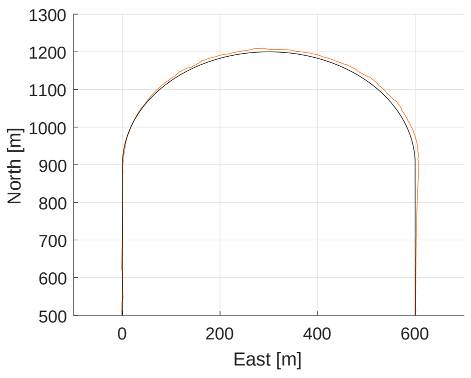
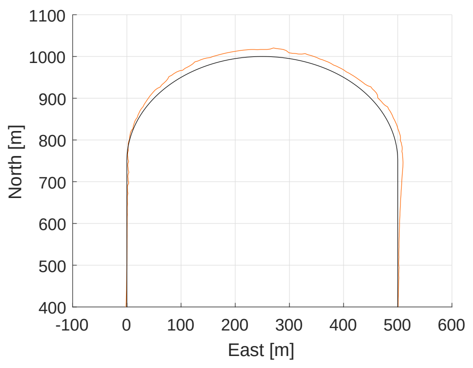


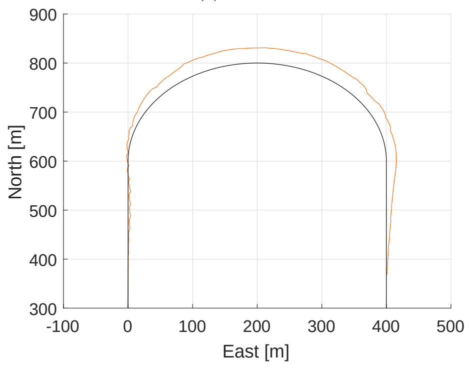
Figure 6.12: The position of the UAV when optimizing a curved 180° turn with varying radius.



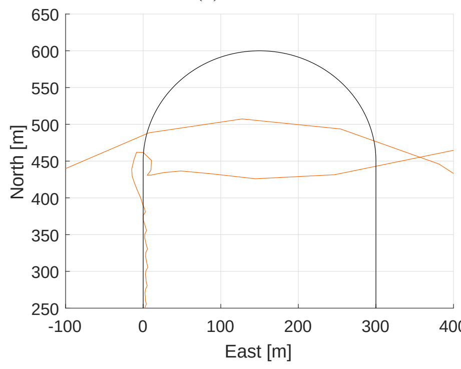
(a) 300m



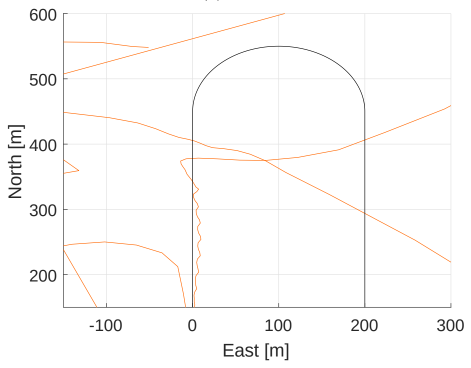
(b) 250m



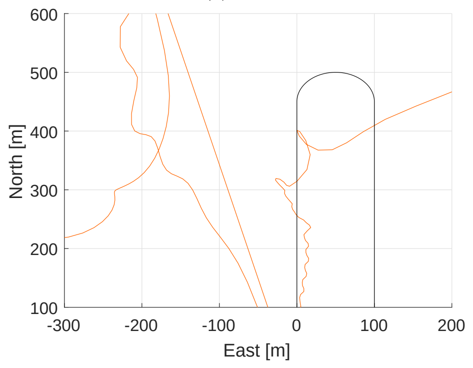
(c) 200m



(d) 150m



(e) 100m



(f) 50m

Figure 6.13: The position of the camera when optimizing a curved 180° turn with varying radius.

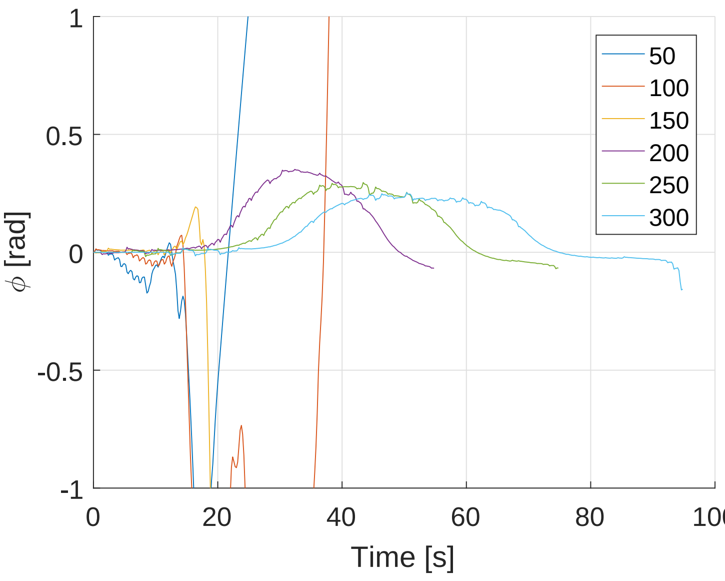
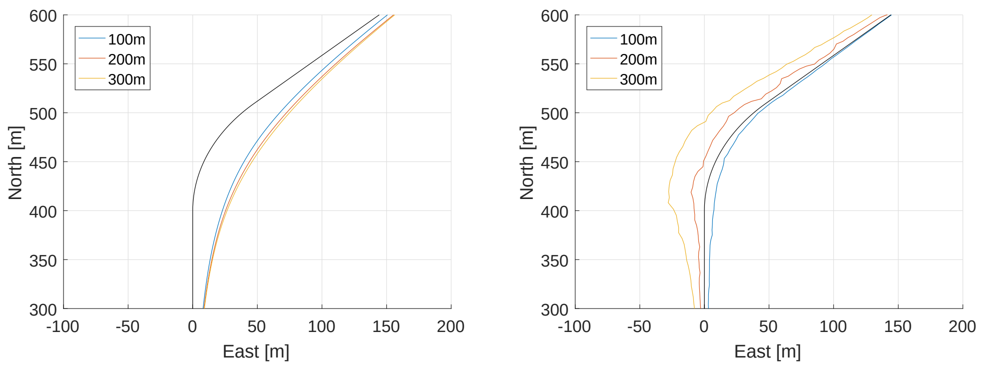


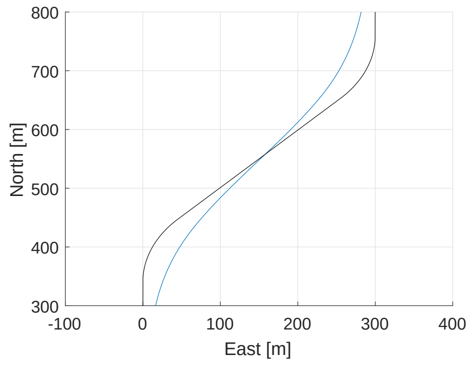
Figure 6.14: The roll angle ϕ during the 180° turns.



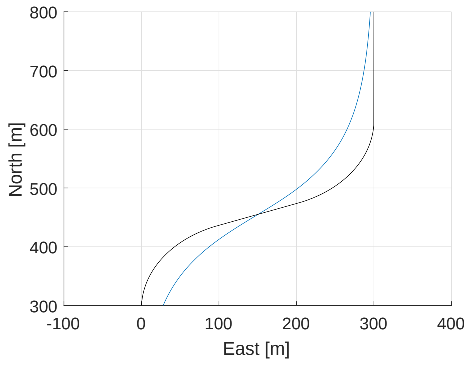
(a) UAV position

(b) Camera position

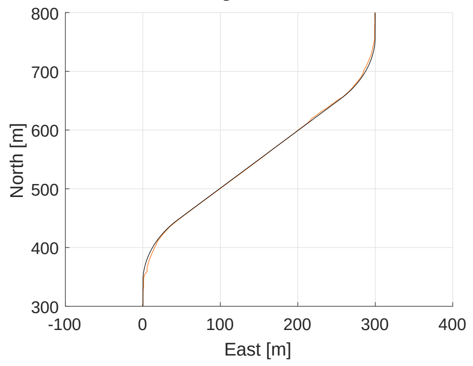
Figure 6.15: The UAV and camera position when tracking a 45° turn with 200m radius at different altitudes.



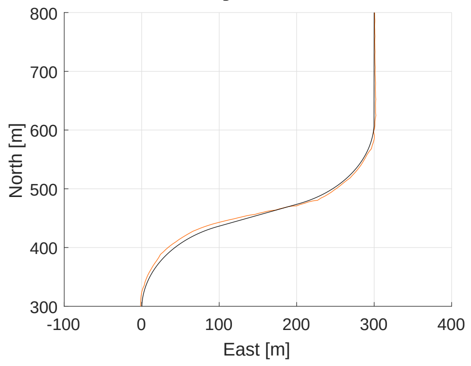
(a) UAV position 45°



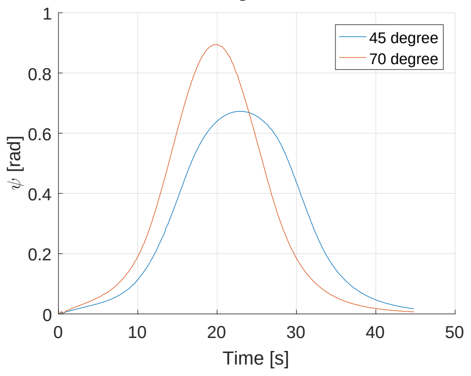
(b) UAV position 70°



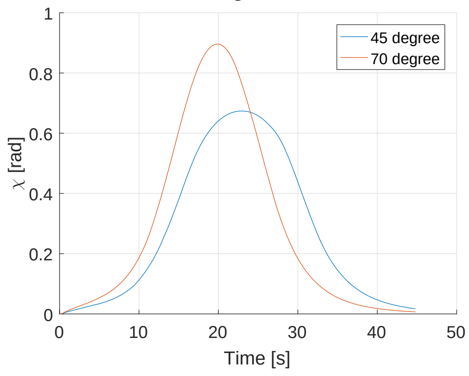
(c) Camera position 45°



(d) Camera position 70°

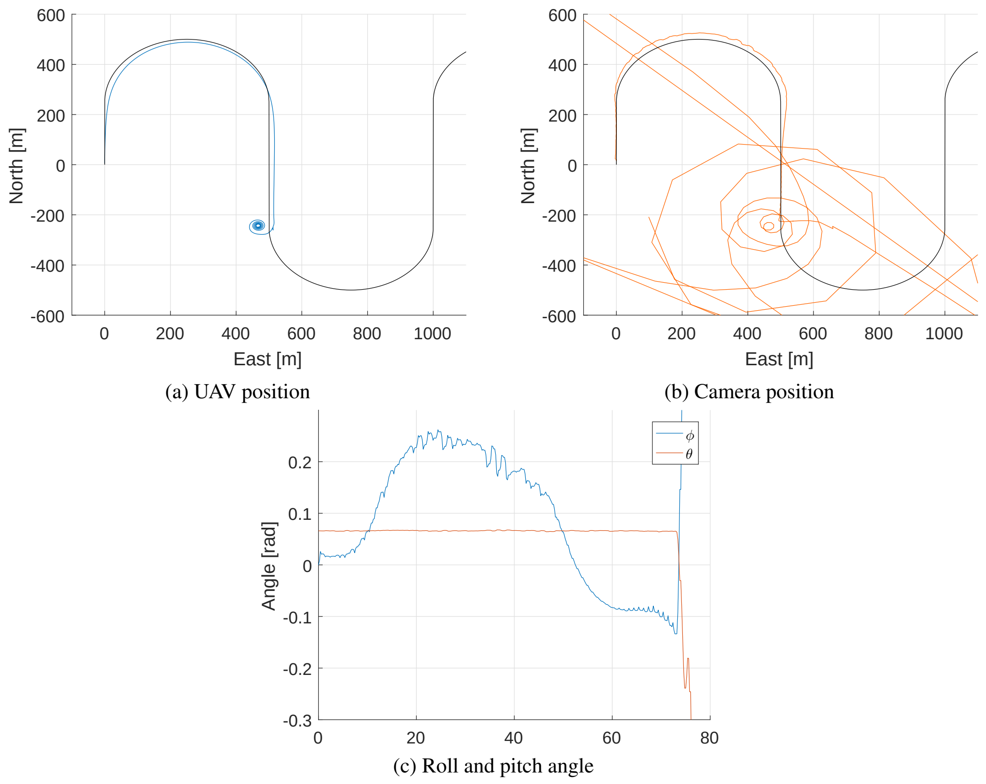


(e) Heading angle



(f) Course angle

Figure 6.16: UAV position, camera position, and heading and course angle during two subsequent turns of 45° and 70°.



(a) UAV position (b) Camera position

(c) Roll and pitch angle

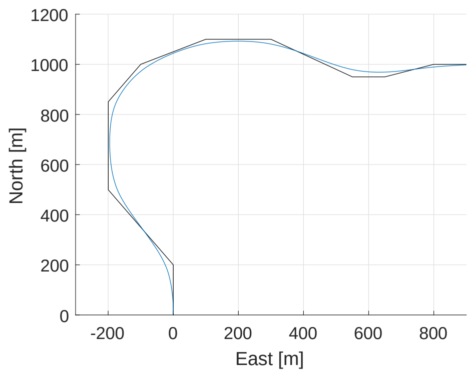
Figure 6.17: Result of attempting to optimize a lawnmover-pattern path with radius 250m.

6.4.2 Lawnmover Path

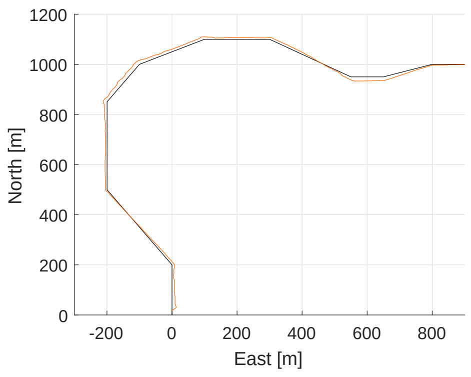
The result of attempting to optimize a lawnmover-pattern path is shown in Figure 6.17. The radius was set to be 250m, as the optimization of 180° turn show that the MPC performs well on this turn, and the line connecting two arcs is set to 500m.

When the MPC optimizes the first 180° turn, it performs in a similar manner to the results in section 6.2.3: it takes the inner turn to begin with, and then widens the turn at the end in order to avoid changing the roll angle quickly. While it should have taken the advantage of the straight line to position the UAV right above the path so it could return to trimmed flight, it instead chooses to continue flying next to the path with a constant roll angle to compensate for the offset from the path. It uses the rudder to compensate for the roll angle so it can travel at a constant course angle. The roll angle it uses to track the path while flying next to the path can be seen in Figure 6.17c.

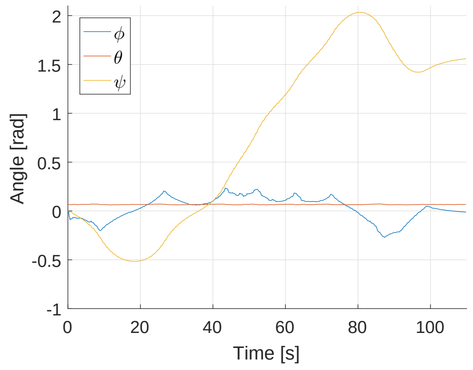
As the UAV approaches the next 180° arch it is already on the inner side of the turn. Instead of this being an advantage, since it previously has optimized a 180° turn like this by taking the inner turn, it throws the UAV into an unstable right-turning spiral.



(a) UAV position



(b) Camera position



(c) Attitude angles

Figure 6.18: Result of optimizing the piecewise linear path.

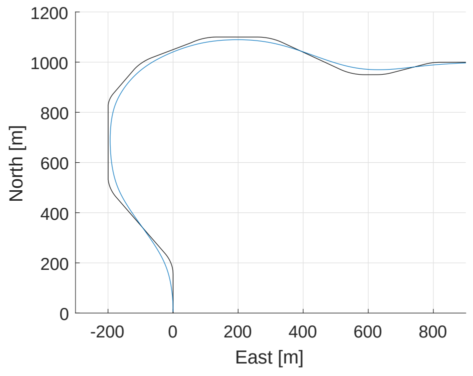
This is similar to previous failures, as the spiral is in the opposite direction of the turn.

6.4.3 Long Paths

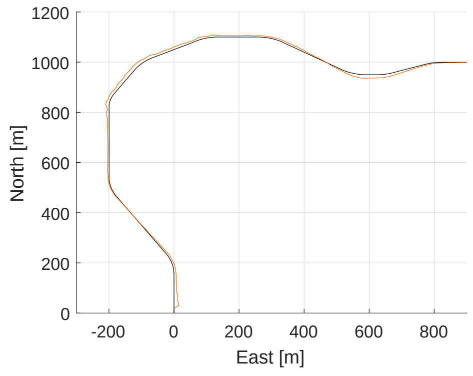
In order to test the performance of the MPC over a longer duration, a longer path consisting of several turns was optimized. The path was designed with linear corners that the MPC could handle, and then the curved path was made with similar waypoints and radius of 150m.

The results of optimizing the longer path can be seen in Figures 6.18 and 6.19, and the results of the two optimizations are very similar. At the beginning of the path the MPC chooses to start the turn immediately, which causes the camera to move quickly to the right for both the linear and the curved path. The long right turn is in both cases optimized by taking one long, continuous turn, with the camera pointing just beside the ground path.

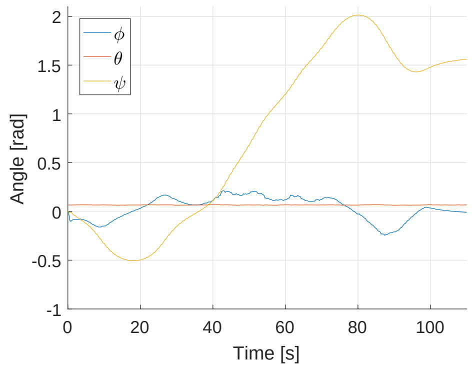
A difference between the linear and the curved paths can be seen in the roll angle in



(a) UAV position



(b) Camera position



(c) Attitude angles

Figure 6.19: Result of optimizing the curved path with a radius of 150m.

	Mean	Max	STD
Linear path	6.8390m	16.8792m	4.1585m
Curved path	5.4001m	14.8386m	3.7833m

Table 6.1: Mean error, max error and the standard deviation between camera centre point and ground path when tracking the path.

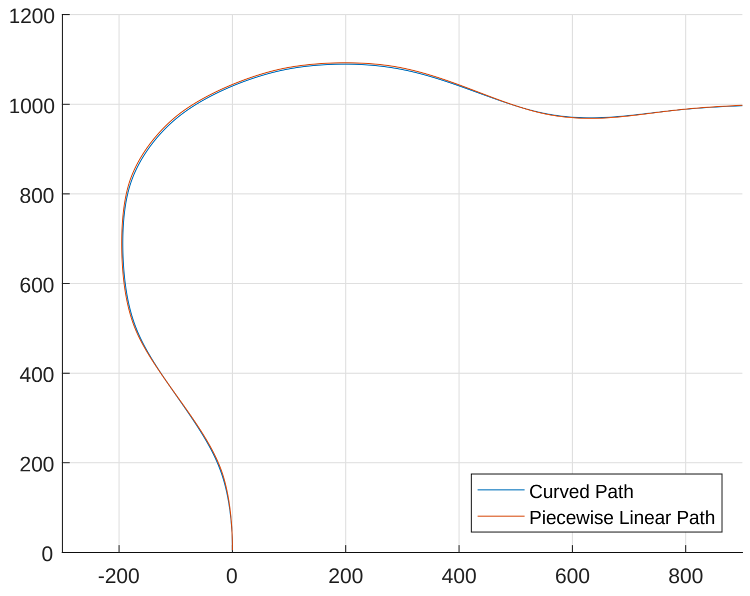


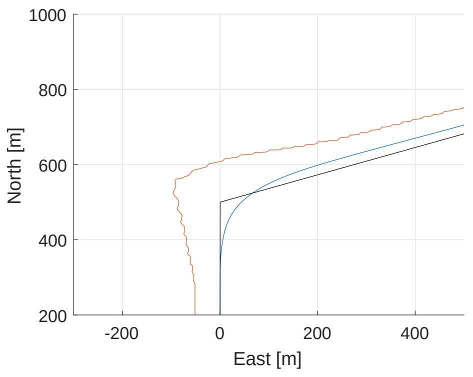
Figure 6.20: The optimized path of the piecewise linear and the curved path.

Figures 6.18c and 6.19c. The peaks in roll are sharper for the linear path than for the curved path, which allows it to better track the sharp corners. Table 6.1 show that the error between the camera centre point and ground path for the linear has both a higher max and a higher mean value than for the curved. The standard deviation is also smaller for the curved path, meaning that there are slightly smaller oscillations when tracking the curved path. In Figure 6.20 the two optimized paths are shown together, and it can be seen that they are almost identical.

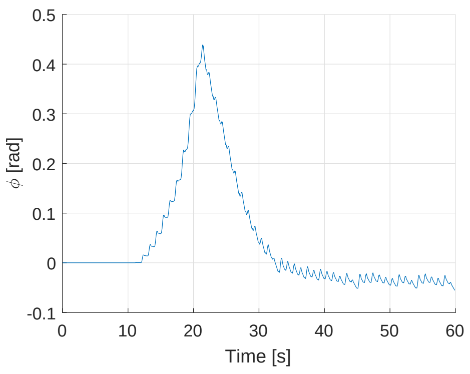
6.5 Reducing the Stepsize

For the results shown in this chapter a stepsize of 0.2s has been used for the MPC. This stepsize returns smooth stable paths for most of the curved paths, but do not work well with the sharper corners in the linear paths. Testing showed that by decreasing the timestep to 0.1s, the MPC manages to return a stable flight path for linear corners with a 70° angle. The MPC was still not able to return a stable flight path for 90° linear turns.

Figure 6.21 shows the result of optimizing a linear 70° turn with a stepsize of 0.1s. Even though the MPC returns a stable flight path, it can be seen that the camera contains many small oscillations. These oscillations can also be seen in the roll angle in Figure 6.21b. The horizon length of 110 timesteps was used for this optimization, but when the stepsize is reduced to 0.1s this is a significant shorter amount of time. Because of this better results might be achieved by increasing the horizon length, when using a



(a) UAV position.



(b) Camera poistion.

Figure 6.21: Results of optimizing a curved 70° turn with reduced stepsize.

stepsize of 0.1s.

Chapter 7

Simulating the Optimized Path

In this chapter paths the SITL simulater described in chapter 5 will be used to fly an optimized path. Ideally the simulated model would be the Aerosonde UAV, as this is the model that has been used as the prediction model in the MPC. However, the ArduPilot SITL interface is only delivered with one default model to interface with JSBSim. This is a model of the Rascal110, and as there was no time to create a model of the Aerosonde the simulations will be performed with this model. Even though the results would have been more precise if the same model was used in both the optimization and the simulation, they will serve to show if the optimized path offer any improvements or not.

As mentioned in chapter 5, the guidance during simulations will be based on the LOS principle. By trial and failure the LOS distance was set to 150m. The path flown by the UAV is shown together with the path that was tracked in Figure 7.1, and the results show that the tracking perfomance of the LOS guidance implemented in DUNE is satisfactory.

7.1 Tracking the Optimized Path

The path that will be tracked is the path that was optimized in section 6.4.3. The optimized paths for both the curved and the piecewise linear path will be tracked. In addition the ground path that is to be observed will be tracked for comparison.

7.1.1 Linear Path

The result of tracking the linear ground path and the optimized path can be seen in Figure 7.2. It is clear that when tracking the ground path the biggest problem is that

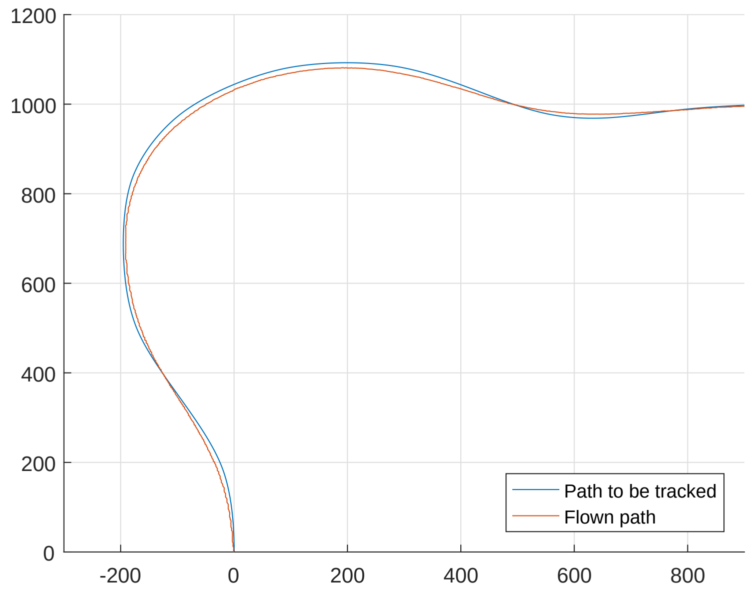


Figure 7.1: Result when tracking the optimized path.

the path that is being tracked consists of linear corners, causing abrupt changes in the roll angle of the aircraft. When tracking the optimized path on the other hand, the are no abrupt changes in the roll angle. The smooth reference path gives a smooth flight path, and the camera centre point stays focused on the ground path without any major deviations.

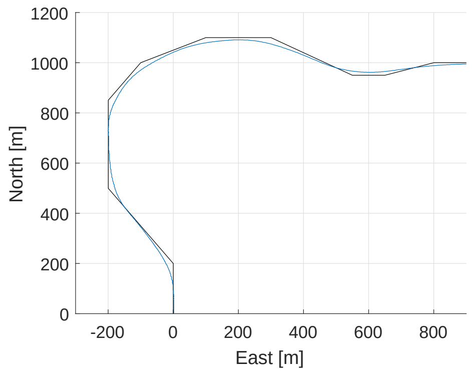
When comparing the attitude of the two simulations, it is clear that the optimized path gives a much more stable flight. Not only do the roll angle vary less, the pitch angle is also close to zero throughout the flight. When tracking the ground path the pitch has some variations, that will further add to movement in the camera position.

7.1.2 Curved Path

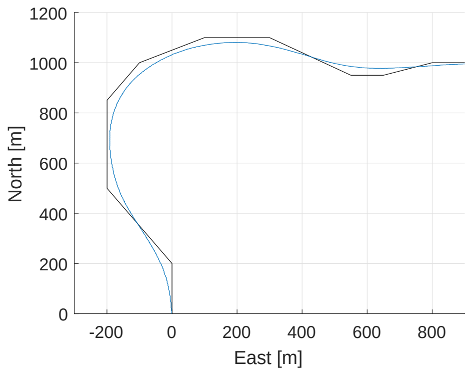
The results for tracking the optimized curved path is very similar to the results for the optimized linear tracking. The biggest difference is that when tracking the curved ground path, the oscillations are smaller than for the linear ground path. The oscillations for the curved optimized path are also slightly smaller than for the optimized linear path, but this difference is so small that it has no practical significance.

7.1.3 Linear VS. Curved

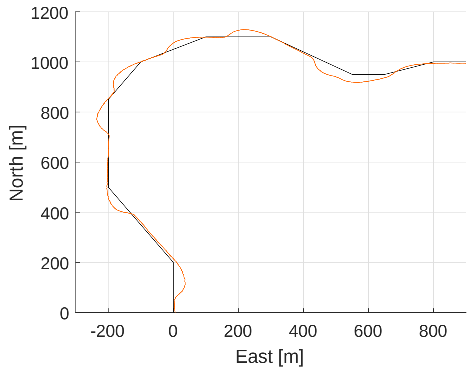
The statistics for the simulations of the paths can be seen in Table 7.1, and they show that the performance when tracking the optimized linear path is better than when track-



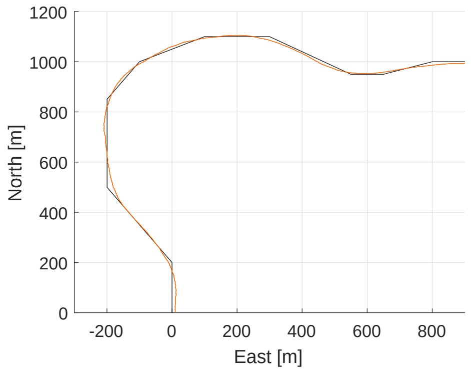
(a) UAV position when tracking ground path



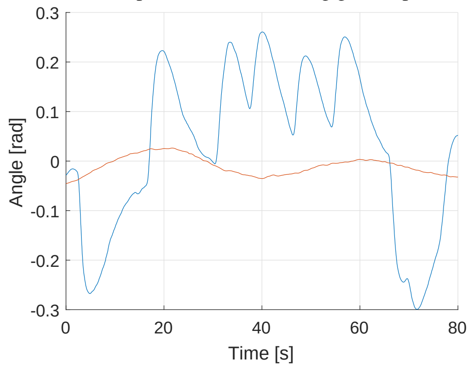
(b) UAV position when tracking optimal path



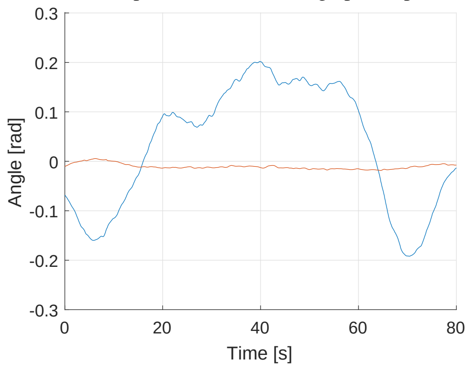
(c) Camera position when tracking ground path



(d) Camera position when tracking optimal path

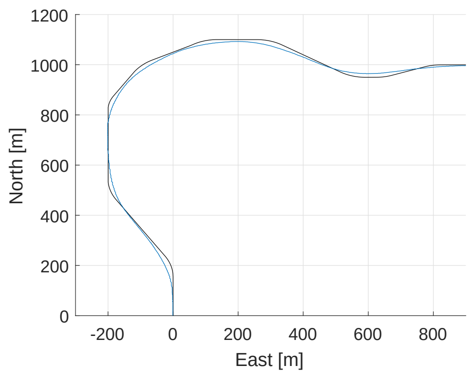


(e) Attitude angles when tracking ground path

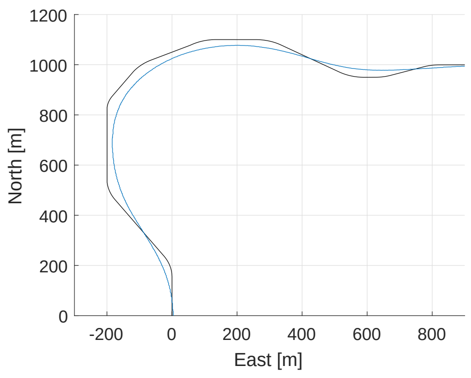


(f) Attitude angles when tracking optimal path

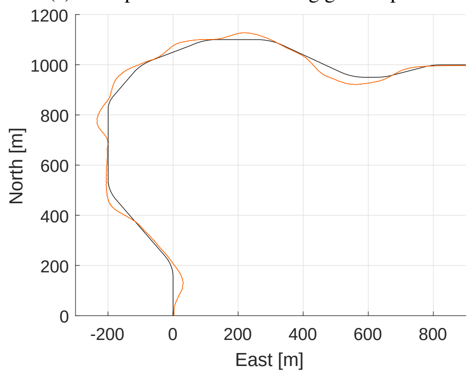
Figure 7.2: The result when tracking the ground path and the optimized path.



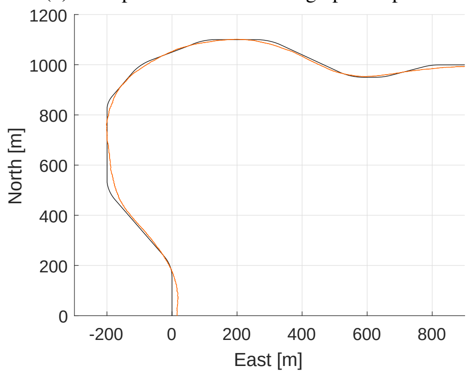
(a) UAV position when tracking ground path



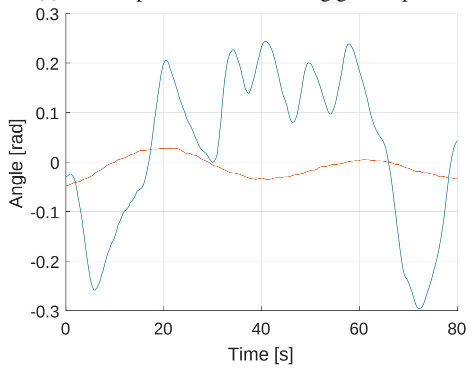
(b) UAV position when tracking optimal path



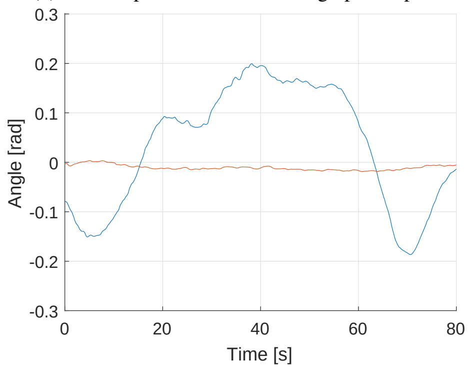
(c) Camera position when tracking ground path



(d) Camera position when tracking optimal path



(e) Attitude angles when tracking ground path



(f) Attitude angles when tracking optimal path

Figure 7.3: The result when tracking the ground path and the optimized path.

	Mean	Max	STD
Optimized Curved Path	4.0366m	21.6539m	5.0647m
Optimized Linear Path	3.5139m	18.4791m	4.0691m
Ground Curved Path	8.0423m	36.4857m	10.4101m
Ground Linear Path	7.9658m	40.5172m	11.2268m

Table 7.1: Mean error, max error and the standard deviation between camera centre point and ground path when tracking the path.

ing the optimized curved path. The mean deviance is 0.5m smaller for the linear path than for the curved path, and the maximum deviance when tracking the linear path is a little more than 3m better than for the curved path. This is opposite of the results when optimizing the paths, where the curved path had a better tracking. It is also worth noting that mean error when optimizing the paths was bigger than the mean error when simulating the paths, and tracking the optimized paths give a considerably better result than tracking the ground path.

Chapter 8

Discussion

The results show that optimizing a path to achieve better observation of the ground path while using a fixed camera is a difficult optimization problem to solve. However, the results also show that tracking can be improved by following an optimized path.

8.1 Piecewise Linear Vs. Curved Paths

There was a big difference in how well the MPC handles piecewise linear and curved paths. While it managed to optimize curved turns as sharp as 90° with a 50m radius, it couldn't optimize linear corners sharper than 45°.

The difference in how well it performs is most likely due to the fact that the piecewise linear paths consists of actual corners, which do not give the same smooth change in reference that curved paths do. These linear corners do not represent a path that a fixed-wing UAV would naturally travel. However, since no processing is performed on the path before it is used as a reference to the optimization problem, the optimization algorithm will try to find a solution that do follow these corners. If some more logic was put into the trajectory generation, it would have been possible to smooth out the corners before they were fed to the optimization algorithm, making it more similar to a curved path.

As shown in the results, the MPC is able to optimize a 70° linear corner when the stepsize is reduced from 0.2s to 0.1s. This makes sense with the rapid changing reference, since the MPC is able to respond to rapid changes in reference quicker with a shorter stepsize. However, the application is very time consuming even when using a stepsize of 0.2s, and decreasing the stepsize would increase this duration, as both the number of intervals and number of steps in the horizon would need to be increased to achieve good results.

A more surprising result when comparing piecewise linear and curved paths is that the linear paths gave a more precise results during simulation. As was seen when comparing the optimization result for curved and linear paths, the linear paths required sharper changes in the roll angle. This information is not embedded in the optimized path, and thus it was not expected that tracking the optimized path would give a better result in the linear case.

8.2 Oscillations (Working Title)

The results in section 6.5 show that when the stepsize is reduced, the oscillations in the states increase. This phenomenon is also present when using a stepsize of 0.2s, but because of the longer time between every re-initiation the effect is not as visible as for 0.1s.

A major contributor to generating these oscillations is the trajectory generator described in section 4.3.1, that assumes the UAV will maintain a fixed speed in order to calculate the distance the UAV will travel during one timestep. This assumption will introduce some inaccuracies, but the results show that during the optimization the speed did not vary much. However, this method of generating the trajectory introduces another inaccuracy, in that the path given as an input is discretized.

Since the reference path is discretized, it is not possible to always find a point up ahead that is exactly the distance the UAV will travel during one timestep away. For this reason, points with a distance away that is within a given range is accepted as the next waypoint. If this range is too big the inaccuracy will be too big, which will cause a spike in the reference model in the cost function. Because of this spike in reference the optimization algorithm will seek to follow the spike, which causes the oscillations in the states.

In Figure 8.1 the effect the acceptance range of waypoints has on the roll through a 45° linear turn is shown. When the acceptance range is ± 0.5m, the magnitude of the oscillations is bigger than for an acceptance range of ± 0.1m. A part from a spike in the wrong direction for the ± 0.5m signal, the two roll angles follow approximately the same trajectory.

8.3 Cost Function

The cost function in this thesis consisted of eight states, where the position of the camera were the states included to solve the problem of finding an optimal path to track with a fixed camera. While this cost function do solve the problem, it does have some problems related to the tuning of it.

Early in testing, before the final tuning of the cost function had been decided, it was clear that the easiest way to solve the control problem was to control the course of the

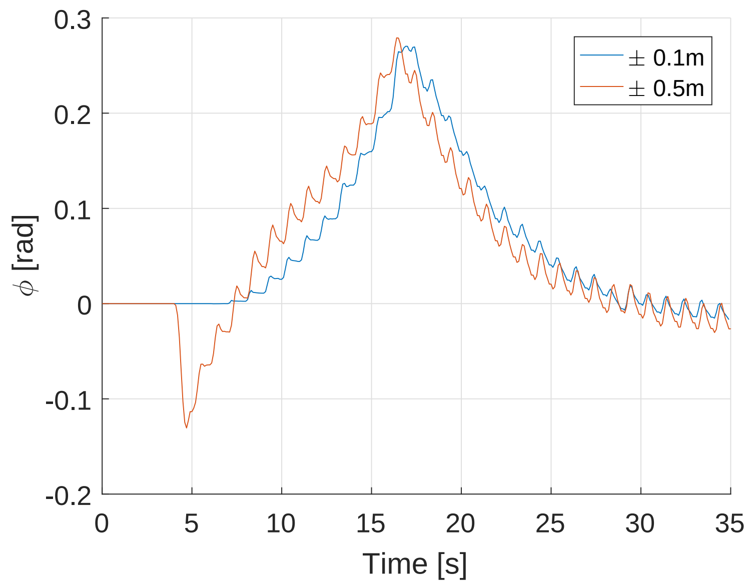


Figure 8.1: The roll angle of the aircraft during a linear 45° turn with different acceptance rates for waypoints.

airplane using the rudder. This is an easy way to the problem since it does not rely on using the roll angle that inherently shifts the camera position. Since this thesis aimed to create an optimal path that could be used with a bank-to-turn controller, this behaviour had to be removed. This was done be heavily punishing usage of rudder, while there was almost no weight put on the usage of ailerons.

Weighting the usage of rudder more than aileron did as intended: The MPC uses roll to follow the path instead of yaw. However, the light weighting of aileron input caused it to oscillate heavily, adding to the effect of the poor trajectory generation previously mentioned.

Another problem discovered during testing was how the MPC deals with long stretches of path. When the turns were positioned closely after one another it was no problem, as the MPC would have to change the course of the aircraft often. When optimizing the lawnmover path in section 6.4.2 however, it became clear that the MPC chose to position the camera centre point on the ground path by keeping a constant roll angle and flying next to the path. Since the cost function only seek to minimize the control rates, this was a cheap solution that required no change of the control states.

One possible solution to this would be to include the position of the UAV in the cost function. These terms would have to be weighted lower than the position of the camera since the camera position is the main focus, but it would cause flying next to the path a more expensive solution compared to flying just above the path. Another solution may be to include control surface deflection in the cost function, but this is most likely a bad decision as some situations require a constant deflection of the control surfaces.

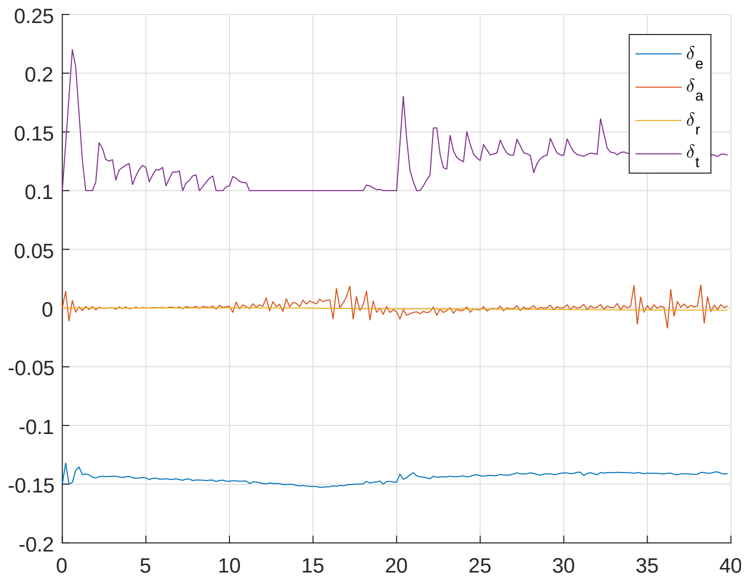


Figure 8.2: The control signals when optimizing a 45° curved turn.

8.4 Control Signals

In this report no plots of the control signals have been given. The main reason for this is that the output of the MPC is the optimized path, and the control signals are not intended to be used for tracking the path. Another reason can be seen in Figure 8.2: they do not always make sense. There are two reasons for this.

Firs of all, the noise is most likely related to the oscillations previously mentioned in this chapter. For every iteration of the MPC there is a significant spike in all of the four control signals, some more than others. Since there is a spike in the reference for every iteration, it is no surprise there is a spike in the control signals with that spike.

The second reason is that the control signals for the aileron and rudder are surprisingly small, and the aileron control signal often do not match the roll angle of the aircraft. The reason for this is most likely tied to the linearized model. The model is already an approximation of the real model, and poor selection of parameters may make matters worse.

Since the UAV states seem reasonable, it was decided not to give much weight to the control signals in this thesis.

Chapter 9

Conclusion

The goal of this thesis was to develop a control method that would create a flight path that would enable a UAV with a hyperspectral camera fixed to its body to optimally survey a ground path. This was solved by creating a intervalwise hyperspectral MPC that seeks to minimize the distance between the camera point on the ground and the ground path that is to be observed. The MPC was implemented using C++ and the ACADO Toolkit.

The results show that the optimization method can be used to achieve good tracking of the ground path. Simulations show that for even a simple path, the mean error when tracking a path can be cut in half when tracking the optimal path compared to tracking the ground path. In addition tracking the optimized path results in less movement of the camera footprint, giving less movement in the captured images. The MPC is able to optimize both curved and piecewise linear paths, with piecewise linear paths giving the most precise end result.

Even thoguh the piecewise linear paths resulted in the most precise tracking, the MPC had a hard time optimizing piecewise linear paths. The sharpest linear corner it could optimize well was a 45° turn. For curved turns it managed to optimize both 90° turn and 180° turns, given a radius larger than 200m. On the other hand, straight paths after turns proved to be a difficulty for the MPC, as it does not correct the UAV position to be straight above the path during these stretches.

Testing showed that the optimization problem was a difficult and time consuming problem to solve. The application created here was not very robust, and it was especially vulnerable to how precise the ground path was given. Testing show that there is most likely an improvement to gain by reducing the stepsize of the MPC from 0.2s to 0.2s however, since the application was so time consuming, it was not thoroughly tested in this thesis.

9.1 Future Work

Testing shows that one of the major shortcomings of the MPC developed in this thesis is how the trajectory is generated for each iteration of the MPC. The method currently used is not precise enough, which leads to spikes in the reference used to solve the optimization problem. If the trajectory generation is improved it would hopefully remove many of the oscillations seen in these results, and make it possible to optimize even sharper corners. A more sophisticated trajectory generator could also allow the MPC to support loitering above a point.

Another problem area for the MPC that should be improved is the duration it takes to optimize the paths. Even though this application isn't intended to run in realtime, it uses what should be an unnecessary amount of time. The ACADO Toolkit offers a code-generation tool, that generates C++ code to be run instead of ”re-compiling” the optimization problem for every iteration of the MPC. Using the code-generation tool could improve the time problem.

Since the MPC already uses a linear model of the UAV, the cost function could also be linearized in order to get a problem that takes less time to solve. Since the ACADO Toolkit is not optimal for linear problems, the linearized problem should be implemented with a different toolkit intended for solving linear problems.

In order to achieve more precise optimization, the complete nonlinear model of the UAV should be implemented. It was attempted in this thesis, but eventually given up on because of time restriction. With more time this should be fully possible to achieve. With the complete nonlinear model it is also possible to include wind in the optimization problem. And lastly, it should be explored including changing altitudes in the optimization problem.

Appendices

Appendix A

UAV Model

A.1 Nonlinear State Space Model

The complete state space equations for the nonlinear model presented by Beard & McLain [14], that is the basis for the linearized model used as the prediction model in the MPC, is given here.

$$\begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta c_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{A.1}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{\mathsf{m}} \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \tag{A.2}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{A.3}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \Gamma_1 pq - \Gamma_2 qr \\ \Gamma_5 pr - \Gamma_6 (p^2 - r^2) \\ \Gamma_7 pq - \Gamma_1 qr \end{bmatrix} + \begin{bmatrix} \Gamma_3 l + \Gamma_4 n \\ \frac{1}{J_y} m \\ \Gamma_4 l + \Gamma_8 n \end{bmatrix} \tag{A.4}$$

f_x, f_y and f_z in (A.2) represent the forces in each direction in the body frame, and m represent the mass of the UAV. In (A.4) the Γ values represent the inertia of the UAV, and l, m, and n is the moments about the axes in the body frame.

A.2 Aerodynamic Force Coefficients

What aerodynamic force coefficients to use with the linearized model is not explicitly stated by Beard & McLain [14]. For this thesis, the linear equations for the coefficients given by Beard & McLain have been used together with the trimmed state values of the UAV states.

$$C_L(\alpha^*) = C_{L_0} + C_{L_\alpha}\alpha \tag{A.5a}$$

$$C_D(\alpha^*) = C_{D_0} + C_{D_\alpha}\alpha \tag{A.5b}$$

$$C_{X_0} = 0 \tag{A.6a}$$

$$C_{X_\alpha}(\alpha^*) = -C_D(\alpha^*)cos(\alpha^*) + C_L(\alpha^*)sin(\alpha^*) \tag{A.6b}$$

$$C_{X_q}(\alpha^*) = -C_{D_q}cos(\alpha^*) + C_{L_q}sin(\alpha^*) \tag{A.6c}$$

$$C_{X_{\delta_e}}(\alpha^*) = -C_{D_{\delta_e}}cos(\alpha^*) + C_{L_{\delta_e}}sin(\alpha^*) \tag{A.6d}$$

$$C_{Z_0} = 0 \tag{A.7a}$$

$$C_{Z_\alpha}(\alpha^*) = -C_D(\alpha^*)sin(\alpha^*) - C_L(\alpha^*)cos(\alpha^*) \tag{A.7b}$$

$$C_{Z_q}(\alpha^*) = -C_{D_q}sin(\alpha^*) - C_{L_q}cos(\alpha^*) \tag{A.7c}$$

$$C_{Z_{\delta_e}}(\alpha^*) = -C_{D_{\delta_e}}sin(\alpha^*) - C_{L_{\delta_e}}cos(\alpha^*) \tag{A.7d}$$

Appendix B

Parameters

In this chapter the different parameters used will be given.

B.1 Trimmed States

States		Control	
p_N	0.0	δ_e	-0.0594
p_E	0.0	δ_a	0.0
h	150	δ_r	0.0
u	25.0	δ_t	0.15
v	0.0	$\Delta\delta_e$	0.0
w	0.0	$\Delta\delta_a$	0.0
ϕ	0.0	$\Delta\delta_r$	0.0
θ	0.066	$\Delta\delta_t$	0.0
ψ	0.0		
p	0.0		
q	0.0		
r	0.0		

Table B.1: The values for the trimmed states used in this paper.

Variable	Min	Max
θ	$-\pi/2$	$\pi/2$
ϕ	$-\pi/2$	$\pi/2$
δ_e	$-\pi/6$	$\pi/6$
δ_a	$-\pi/6$	$\pi/6$
δ_r	$-\pi/6$	$\pi/6$
δ_t	$-\pi/6$	$\pi/6$
$\Delta\delta_e$	-0.2	0.2
$\Delta\delta_a$	-0.2	0.2
$\Delta\delta_r$	-0.2	0.2
$\Delta\delta_t$	-0.2	0.2

Table B.2: Constraints used in the optimization problem.

B.2 Optimization Parameters

MPC		Cost Function	
Horizon length	110	Q(p_N)	10^{-1}
Timestep	0.2s	Q(p_E)	10^{-1}
Interval length	5	Q(u)	10
Acceptance range trajectory	$\pm 0.08m$	Q(h)	10
		Q($\Delta\delta_e$)	10
		Q($\Delta\delta_a$)	10^{-1}
		Q($\Delta\delta_r$)	10^{5}
		Q($\Delta\delta_t$)	10^{0}

Table B.3: Tuning parameters used for the optimization problem.

Appendix C

MPC Algorithms

Algorithm 1 Offline Intervalwise MPC Algorithm

```
procedure MPC
    path ← path from file
    timestep ← duration of timestep [s]
    horizonlen ← number of timestep in horizon
    intervallen ← number of timestep in interval
    intervals ← number of intervals needed to cover path
    $x_0$ ← initial values of states
    $u_0$ ← initial values of control states
    $\Delta u_0$ ← inital values of control rates
    results[] ← empty list to store result from optimization
    for each interval do
        $\mathbf{c}^n$ ← calculate camera centre position using equation 2.14
        trajectory ← GenerateHorizon(path, timestep, horizonlen, $\mathbf{c}^n$)
        Solve optimization with initial states $x_0$, $u_0$, $\Delta u_0$ for current horizon
        $x_0$ ← last $x$ value in the interval
        $u_0$ ← last $u$ value in the interval
        $\Delta u_0$ ← last $\Delta u$ value in the interval
        result[] ← the first intervallen number of timesteps from horizon
```

Algorithm 2 Generate horizon

```
procedure GenerateHorizon(path, timestep, horizonlen, $\mathbf{c}^n$)
    distance ← distance travelled during one timestep
    pos ← find the point in path that is closes to current camera position $\mathbf{c}^n$
    trajectory[] ← empty list to store the generated trajectory
    for each timestep in horizonlen do
        Find point $pos_{temp}$ on path with the given distance away from current pos
        trajectory[] ← $pos_{temp}$
        pos ← $pos_{temp}$
    return trajectory
```

Bibliography

[1] W. H Kwon and S. H. Han. *Receding Horizon Control: Model Predictive Control for State Models*. Advanced Textbooks in Control and Signal Processing. Springer London, 2005.

[2] E. Skjong, S. A. Nundal, F. S. Leira, and T. A. Johansen. 2015 international conference on unmanned aircraft systems (ICUAS). In *Autonomous Search and Tracking of Objects Using Model Predictive Control of Unmanned Aerial Vehicle and Gimbal: Hardware-in-the-loop Simulation of Payload and Avionics*, Denver, Colorado, USA, June 2015. IEEE.

[3] J. Egbert and R. W. Beard. Proceedings of the 2007 American control conference. In *Low Altitude Road Following Constraints Using Strap-Down EO Cameras on Miniature Air Vehicles*, New York City, USA, July 2007. IEEE.

[4] Thomas M. Fisher. Rudder augmented trajectory correction for unmanned aerial vehicles to decrease lateral image errors of fixed camera payloads. *All Graduate Theses and Dissertations*, 2016.

[5] S. Mills, J. J. Ford, and L. Mejias. Vision based control for fixed wing UAVs inspecting locally linear infrastructure using skid-to-turn maneuvers. *Journal of Intelligent and Robotic Systems*, 61(1):29–42, 2011.

[6] M. Ahsan, H. Rafique, and Z. Abbas. Multitopic conference (INMIC). In *Heading Control of a Fixed Wing UAV Using Alternate Control Surfaces*. IEEE, December 2012.

[7] Stephen P. Jackson. Controlling small fixed wing UAVs to optimize image quality from on-board cameras. *ProQuest Dissertations and Theses*, 2011.

[8] Randall B. Smith. *Introduction to Hyperspectral Imaging*. MicroImages, Inc., 2012.

[9] R. Nsi, E. Honkavaara, P. Lyytikinen-Saarenmaa, M. Blomqvist, P. Litkey, T. Hakala, N. Viljanen, T. Kantola, T. Tanhuanp, and M. Holopainen. Using

UAV-based photogrammetry and hyperspectral imaging for mapping bark beetle damage at tree-level. *Remote Sensing*, 7(15467-15493), 2015.

[10] P. J. Zarco-Tejada, V. Gonzlez-Dugo, and J. A. J. Berni. Fluorescence, temperature and narrow-band indices acquired from a UAV platform for water stress detection using a micro-hyperspectral imager and a thermal camera. *Remote Sensing of Environment*, 117(322-337), 2012.

[11] J. Suomalainen, N. Anders, S. Iqbal, G. Roerink, J. Franke, P. Wenting, D. Hnniger, H. Bartholomeus, R. Becker, and L. Kooistra. A lightweight hyperspectral mapping system and photogrammetric processing chain for unmanned aerial vehicles. *Remote Sensing*, 6(11013-11030), 2014.

[12] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd, 2011.

[13] Guowei Cai, Ben M. Chen, and Tong Heng Lee. *Unmanned Rotorcraft Systems*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[14] Randal W. Beard and Timothy W. McLain. *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, Princeton, NJ, USA, 2012.

[15] O. Egeland and J. T. Gravdahl. *Modeling and Simulation for Automatic Control*. Marine Cybernetics, Trondheim, Norway, 2002.

[16] E. F. Camacho and Bordons C. *Model Predictive Control*. Springer London, 1999.

[17] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[18] M. Diehl, H.G. Bock, H. Diedam, and P.-B. Wieber. *Fast Direct Multiple Shooting Algorithms for Optimal Robot Control*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[19] S. H. Mathisen, T. I. Fossen, and T. A. Johansen. Non-linear model predictive control for guidance of a fixed-wing uav in precision deep stall landing. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 356–365, June 2015.

[20] J. B. Rawlings and D. Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, Madison, Wisconsin, 2015.

[21] B. Houska, H.J. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[22] TD Bui and TR Bui. Numerical methods for extremely stiff systems of ordinary differential equations. *Applied Mathematical Modelling*, 3(5):355–358, 1979.

[23] Facculty of Engineering University of Porto. LSTS: DUNE Unified Navigation Environment. http://www.lsts.pt/toolchain/dune. Accessed: 23.05.2017.

[24] ArduPilot. http://ardupilot.org/. Accessed: 23.05.2017.

[25] JSBSim. http://jsbsim.sourceforge.net/index.html. Accessed: 23.05.2017.

[26] Facculty of Engineering University of Porto. Neptus. http://www.lsts.pt/toolchain/neptus. Accessed: 23.05.2017.

[27] K. Gryte and T. I. Fossen. *High Angle of Attack Landing of an Unmanned Aerial Vehicle*. Norges Teknisk-Naturvitenskapelige Universitet, Trondheim, Norway, 2015.

[28] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[29] A. Asmat, E.J. Milton, and P.M. Atkinson. Empirical correction of multiple flightline hyperspectral aerial image mosaics. *Remote Sensing of Environment*, 115(10):2664 – 2673, 2011.