NTNU
Norwegian University of
Science and Technology

Model predictive waypoint following for an UAV using end-time bisection

Nikolai Mejdell Hektoen

Master of Science in Cybernetics and Robotics
Submission date: Januar 2016
Supervisor: Lars Imsland, ITK
Co-supervisor: Anders Albert, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Faculty of Information Techonology,
Mathematics and Electrical Engineering

Department of Engineering Cybernetics

MASTER'S THESIS

Name of the candidate:	Nikolai Mejdell Hektoen
Subject:	Engineering Cybernetics
Thesis title:	Model predictive waypoint following for an UAV using end-time bisection

Background

Unmanned Aerial Vehicles (UAVs) are becoming an increasingly popular research subject. The reason is the many applications UAVs can perform. Examples of tasks are surveillance, target tracking, search and rescue, and communication relay. One application that has become highly relevant is iceberg tracking. Icebergs can pose a threat for operations such as oil exploration, production, and transport. Surveillance operations often use satellites. However, the coverage of the northern area, where icebergs are present, is scarce and it is very expensive to increase the coverage. UAVs are cheaper in use, operators can better control them and they are closer to the ground. This enables better and faster coverage of areas of interest.

One of the advantages with an UAV is that it is possible to define a path using an optimization algorithm based on certain set of criteria, compared to satellites that must follow a predefined path. However, to follow a defined path the UAV needs a feedback controller. An optimization algorithm may describe a path with waypoints, 2D coordinates or 2D coordinates with heading and actuator input. A feedback controller must use the defined path and state of the UAV to define the actuator input for the UAV. This project is related to the use of nonlinear model predictive control for solving the task of path following. It is proposed that the work is done using the CasADi framework in e.g. Python. The project builds on work done in specialization project spring 2015.

Tasks:

1) Give a brief background on control methods for path (waypoint) following
2) Give a brief introduction to dynamic optimization with emphasis on collocation methods
3) Extend/improve waypoint path optimization from project work
4) Use this to design an MPC waypoint following controller
5) Test and evaluate the controller in various scenarios (nominal, with disturbances (wind), model/plant mismatch, etc.)
6) Discuss, conclude, suggest further work

Supervisor: Professor Lars Imsland,

Co-supervisor: Anders Albert (PhD-candidate)

ABSTRACT

Ice surveillance is motivated by a need to improve the safty for shipping and offshore industries in the arctic waters. Unmanned aerial vehicles (UAVs) are considered a cost-efficient way of handling ice surveillance. This thesis describes the implementation and simulation results for a model predictive controller (MPC) for a iceberg surveillance by UAVs, using an end-time-based optimization scheme. The optimization scheme uses a bisection method to find the end-time over a series of smaller optimizations. The implementation of the MPC scheme and optimization algorithm was done in Python, using the open source sybolic framework Casadi for algorithmic differentiation.

Generally, the MPC performed well when the wind distubance on the UAV was known, since the wind then could be coupled foreward by the optimal controller. It was found that the UAV could handle wind disturbances of more than half its own speed. However, in cases where the information of the wind disturbance was incomplete, the simulated UAV could be blown off course because there was no integral effect in the control loop, to correct the error. The bisection method used on the end-time in the optimal controller found good solutions as long as the vehicle was not too close to the target waypoints.

Baring these two problems, the whole control system was found to perform adequately although not perfect. By including the suggested wind estimator and backup controller, which addresses the mentioned problems, the overall performance would probably improve.

SAMMENDRAG

Isovervåkning er viktig for å øke sikkerheten rundt oljevirksomhet og transport in arktiske farvann. Ubemannede droner (UAV) er trolig en kostnadseffektiv måte å håndtere denne overvåkninge på. Denne avhandlingen beskriver implementeringen og simuleringsresultatene av en modell prediktiv kontroller (MPC) for en UAV for isfjellovervåkning. Kontrolleren bruker en tidshorisont-basert optimaliseringsalgoritme, der slutttiden for optimaliseringen ble funnet ved hjelp av en bisection-metode over flere små optimaliseringer. Implementeringen av MPC-en og den optimale kontrolleren ble gjort i Python, med det symbolske rammeverket Casadi.

Den modell prediktive kontrolleren fungerte tilfredsstillende så lenge vindforstyrrelsene på UAV-en var kjent, siden den optimale kontrolleren foroverkobler forstyrrelsene. Det ble funnet at UAV-en tålte vindforstyrrelser med hastighet på mer enn halvparten av hastigheten til UAV-en. Kontrolleren fikk imidlertid problemer i tilfeller der deler av vindforstyrrelsen var ukjent. UAV-en kunne blåse ut av kurs, siden det ikke var noen integraleffekt i kontrollsystemet. Den optimale kontrolleren og bisection-metoden fungerte tilfredsstillende for å finne optimale løsninger så lenge UAV-en ikke var for nær neste navigeringspunkt.

Bortsett fra disse to problemene fungerte hele kontrollsystemet godt. Ved å inkludere foreslåtte forbedringer som en vindestimator og en reservekontroller, vil systemet sannsynligvis bli ytterligere forbedret.

PREFACE

This document is my Master's Thesis and is the final work in my degree in Engineering Cybernetics at NTNU. The thesis is the results of work during the fall semester of 2015, and is build upon the results from a project done in the spring of the same year.

I would like to thank my supervisors, Professor Lars Imsland and Ph.D. student Anders Albert for their input and guidance during the work on this thesis and the proir project work. I would also like to thank my parrents for taking their time to read and give feedback on the report.

CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF CODE SNIPPETS

ABBREVIATIONS

UAV	Unmanned Aerial Vehicle
MPC	Model Predictive Control
DOF	Degree of Freedom
NMPC	Nonlinear Model Predictive Control
LOS	Line of Sight
DP	Dynamic Positioning
OPC	Optimal Control Problem
PID	Proportional Integral Derivative (regulator)

1 INTRODUCTION

1.1 BACKGROUND

Unmanned Aerial Vehicles (UAVs), often called drones, are a type of aircrafts, which are controlled without a pilot on board. They are either remotely operated by a pilot on the ground, or they are autonomous, which means they are controlled by a guidance system, without the need of a human pilot. Since UAVs are operated without a pilot on board, they can be built much smaller and lighter than regular aircrafts, and still do the same job. The early history of UAVs consists of mostly of military usages, but in recent years more and more civilian purposes for them are developed, due to the technology becoming cheaper and more accessible.

1.1.1 The history of unmanned aerial vehicles

The first UAVs were no more than flying bombs, with the earliest being unmanned bomb-filled balloons used by the Austrians in an attack on Venice in 1849 [1]. However, the plan was not successful, as some of the balloons blew back across the Austrian lines due to change in wind direction and no possibility for steering. In the early 19th century some UAVs were developed and used as targets for training military personnel. During World War I, various "aerial torpedoes" were invented, but never used. The control mechanism for these were quite simple by today's standards. The planes were programmed to fly a certain distance, then suddenly pitch downwards with the payload, and hopefully hit the intended target. Later, in World War II, many different drones were developed, both for surveillance and combat. The German V-1 [2] rocket was probably the most known and feared of them, and was basically a bomb with wings propelled by a primitive jet engine.

It was not before the Vietnam War that UAVs as we know them today began to emerge. Some were used for dangerous surveillance missions, mainly to save the lives of pilots. In the seventies, the Israeli air force developed the UAV that is regarded as the first modern surveillance UAV. The Tadrian Mastiff [3] could stay airborne for more than 7 hours and had a data-link system that made it possible to stream a live video feed down to the operators. As the technology improved, so did the UAVs, and in 1995, maybe the most well-known UAV of them all was introduced; the Predator drone, which, together with successor Global Hawk, are still the primary drones used by the United States Air Force [2].

In recent years the research on various non-military usages for UAVs has increased. Since UAVs are unmanned they are perfect for tasks where the conditions are too harsh for manned vehicles, where manned vehicles are too large, or there is simply no need for a human pilot in the plane. Today UAVs come in many different sizes and forms, and are used in a great variety of fields. These include small multi-copters used for different kinds for photography, the delivery drone the webstore Amazon are developing [4], and bigger plane-like drones used to inspect crop health in agriculture [5], photographing the receding glaciers on Greenland [6] or the focus of this project: ice drift surveillance.

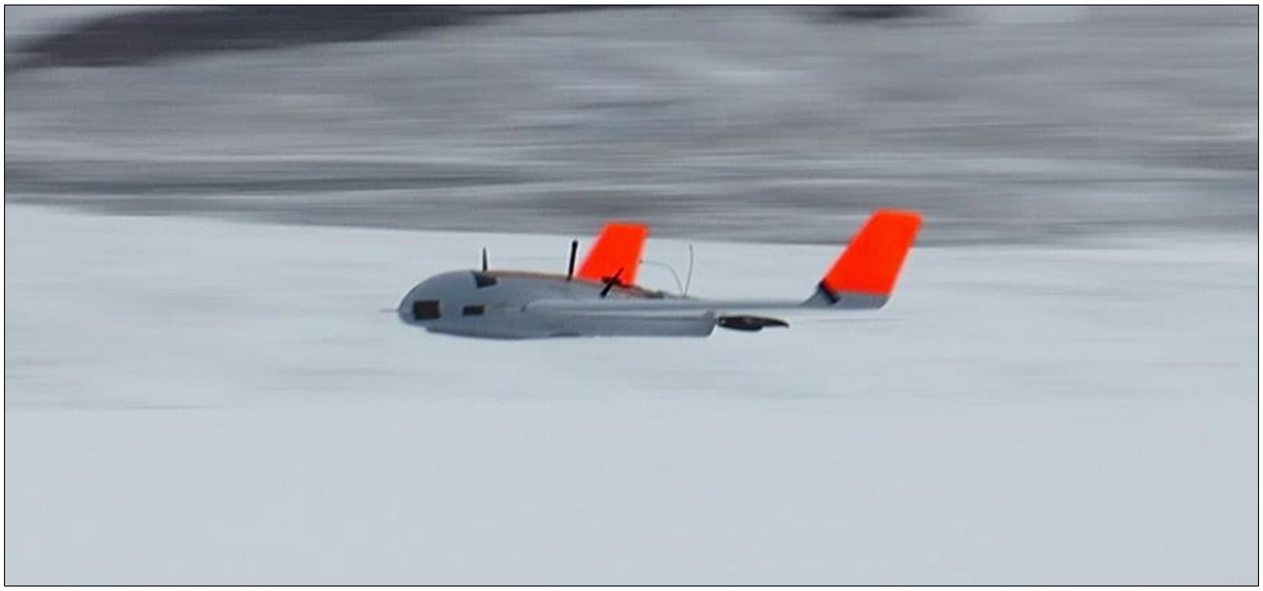

Figure 1-1: The UAV used in flight tests on Svalbard by NTNU (Photo: Pål Kvaløy)

1.1.2 Ice surveillance

The need for ice surveillance is motivated by the desire to utilize the artic sea for shipping and oil drilling. When operating in the Artic Sea, the drifting ice can be a great danger to installations and vessels, therefore it is of vital importance that knowledge of the ice movement is gathered so that the appropriate actions can be taken. Due to different challenges, it is common to distinguish between the surveillance of sea-ice and icebergs [7]. With sea-ice, the focus is mainly on coverage and thickness, while with icebergs the focus is mainly on size and drift direction.

Today, ice surveillance is done using either sensors on the ice itself, satellite images or manual observations from planes and ships. Many of these methods have some challenges. The main disadvantage of satellites is that the satellites have to be in a polar orbit. These orbits are not very suited for targeted surveillance since they only pass over the same area once per day due to the rotation of the Earth [8]. Sensors on the ice itself, while accurate, have to be placed there manually and recovered later when the ice melts.

All these methods have in common that they are quite expensive. Due to the flexibility, speed and low operation cost of UAVs, they could be a useful addition to the methods used today for ice surveillance [9].

1.2 EARLIER WORK

1.2.1 Arctic dynamic positioning project

In December 2014, the research project "Arctic DP: Safe and green dynamic positioning operations of offshore vessels in an Arctic environment"[10] was finished. The project was coordinated by NTNU, with partners such as the Kongsberg Group and Statoil. As concluded in the project, one of the main challenges with dynamic positioning in the Arctic Sea is the ice. In this project an Ice Management System is described which includes 1) icebreakers to break up the drifting ice into smaller parts which the installations can handle, and 2) an automated ice surveillance system which makes it possible to detect and estimate the direction and speed of the drifting ice. This surveillance system can acquire data from a number of different sources, such as satellites, sensors on the ice and both underwater and aerial drones.

1.2.2 Project assignment spring 2015

This project is built upon the previous project work [11] (TTK4550) done by the author in the spring of 2015, in which an optimal controller was developed for an UAV for ice berg surveillance.

The icebergs was represented by a set of waypoints, and the optimal controller was designed to make the UAV fly through each waypoint with its front turned towards the next waypoint. To achieve this, a bisection algorithm, which is an iterative root-finding method, was used on the optimization time to find an optimal path. As shown in Figure 1-2, the optimal path found had a better performance than a Line of Sight (LOS) control algorithm. However, the path generated was not as short as the Dubins path. It was therefore concluded that there was needs for further optimization.

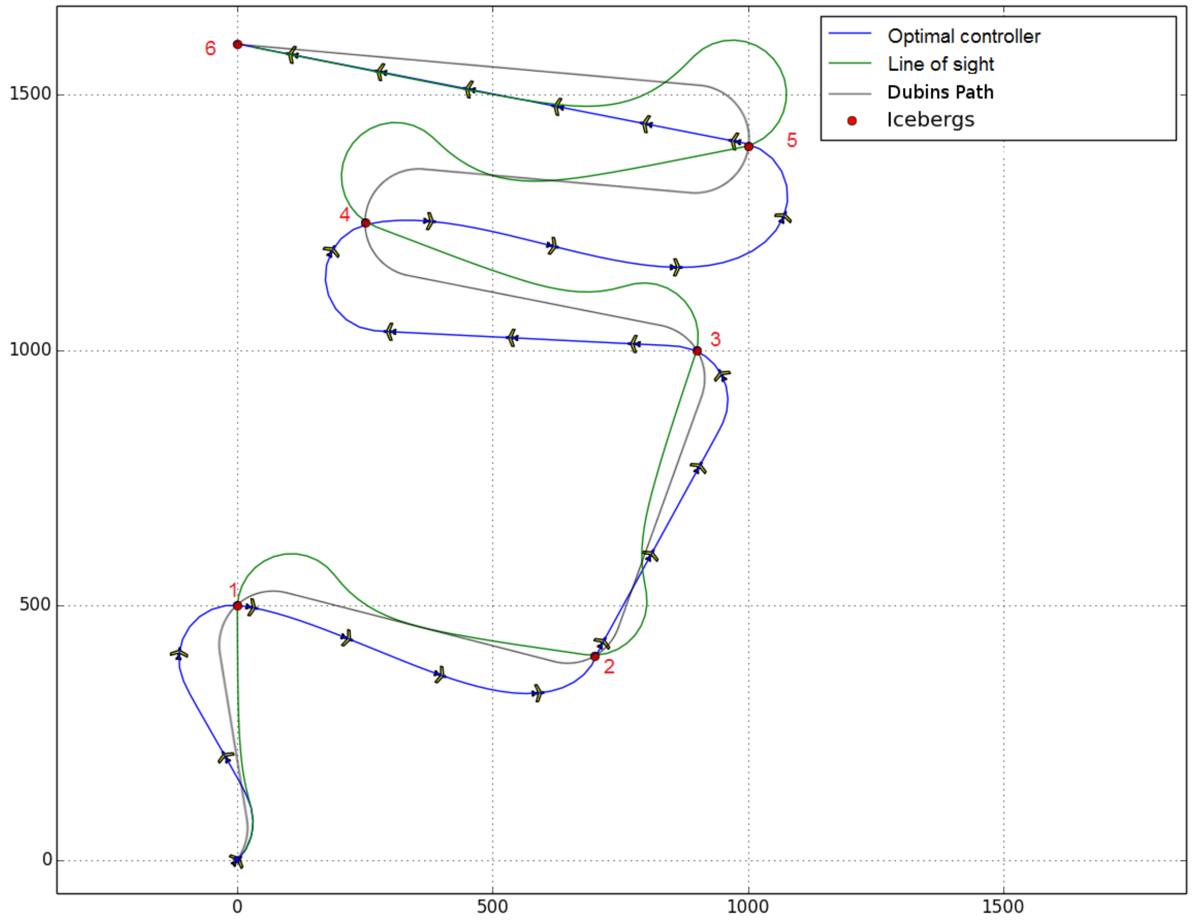

Figure 1-2: Optimal Path planner from project assignment spring 2015.

Table 1-1: Result from project assignment. Length of the path through the waypoints.

	Line of sight	**Optimal controller**	**Dubins Path**
Length (m):	5110 m	5040 m	4565 m
Difference from Dubin:	+545 m	475m	-

In early autumn of 2015, some additional work were done on the objective function and end constraints for the controller and optimization algorithm. This further improved the performance of the optimal controller to match that of the optimal Dubins solution.

1.3 OBJECTIVE OF THE THESIS

The main objective of this project was to design and implement a Model Predictive Controller (MPC) for an UAV using an improved version of the path optimization algorithm from the author's previous work, and further to discuss the performance for this control scheme in various scenarios.

In more detail, the tasks were:

1. Give a brief background on control methods for path (waypoint following).
2. Give a brief introduction to dynamic optimisation with emphasis on collocation methods.
3. Extend/improve waypoint path optimization from project work.
4. Use this to design a MPC waypoint following controller.
5. Test and evaluate the controller in various scenarios (nominal, with disturbances, model/plant mismatch, etc.)

In chapter two, the theory behind waypoints following, optimization and model predictive control is explained. Furthermore there is a section about the bisection method used by the optimal controller and in the end of the chapter, there is a brief description of the software used. Chapter three covers the implementation. Here, the implementation of the optimal controller and the model predictive controller are explained. Chapter four contains the simulations and results. The test scenario is first outlined and then the results from the simulations are shown. The last two chapters covers discussion and conclusions.

2 THEORY

This chapter gives background information of the methods later used in the project. Initially there is a section about Waypoint following, where the Line of Sight algorithm and the concept of Dubins path are explained. Then, nonlinear optimization, optimal control with focus on the direct collocation method, and model predictive control are covered. Lastly, there is a section about the bisection method and a short overview of the software used in the project.

2.1 WAYPOINT FOLLOWING

Waypoint following is the problem of navigating a vehicle through a series of waypoints. The methods used to solve this problem differ in how they use the waypoint to set the course. Some methods navigate using just the waypoints themselves and calculate a course by simple geometry, while other methods divide the problem into a path generation problem and a path following problem. In this project, both the generation- and following problems are solved at the same time using an optimal controller in an MPC-scheme.

2.1.1 Line of sight algorithm

The line of sight algorithm (LOS) is one of the more common methods used for waypoint following [12]. This method finds the target heading by steering towards a point on the straight lines between the waypoints, this point is often placed at fixed distance ahead of the perpendicular line from the vehicle down onto the lines between the waypoints, as seen in Figure 2-1. This method work well in many cases, and with tuning it can perform quite well in comparison with more advanced methods. However, the LOS method does not take into account more than the current waypoint when calculating the course, and leaves therefore room for improvement.

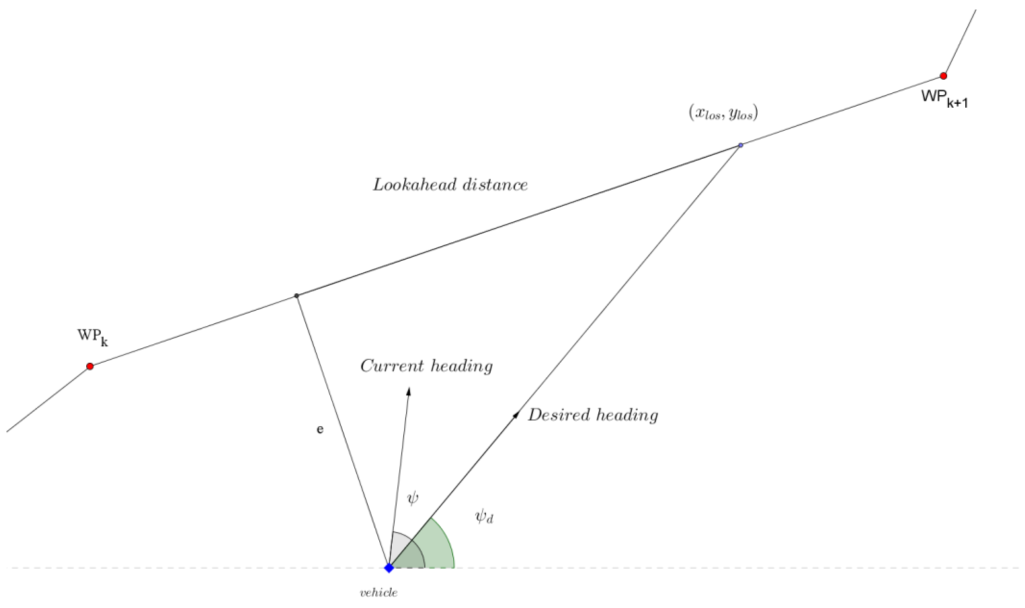

Figure 2-1: Calculation of the desired heading for a vehicle using the Line of sight algorithm.

2.1.2 Dubins optimal path

The mathematician Lester Eli Dubin showed in 1957 [13] that the shortest path through a number of points in a 2D plane, when a limitation is placed on the turn-rate, will be an combination of straight lines and circular arcs with the maximum curvature allowed. Dubins path is commonly used in robotics and control theory, but there are also other methods of finding trajectories which take into account the acceleration of the turn movement such as splines and κ-trajectories [14].

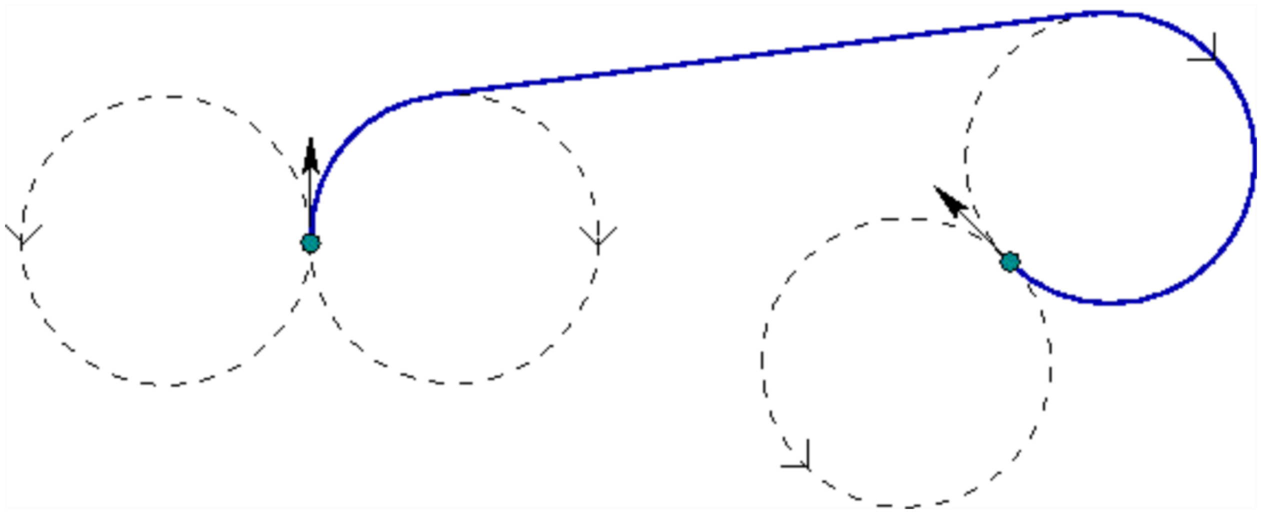

Figure 2-2: Dubins path, consisting of straight lines and circle segments with the maximum turn rate.

Vehicle models with limited turn rate in a 2D plane is often called Dubins Vehicles since the Dubins conditions apply to them. The UAV in this project was simplified into a Dubins Vehicle by removing the altitude from the equation, making a system model consisting of just the north- and east positions and the heading angle.

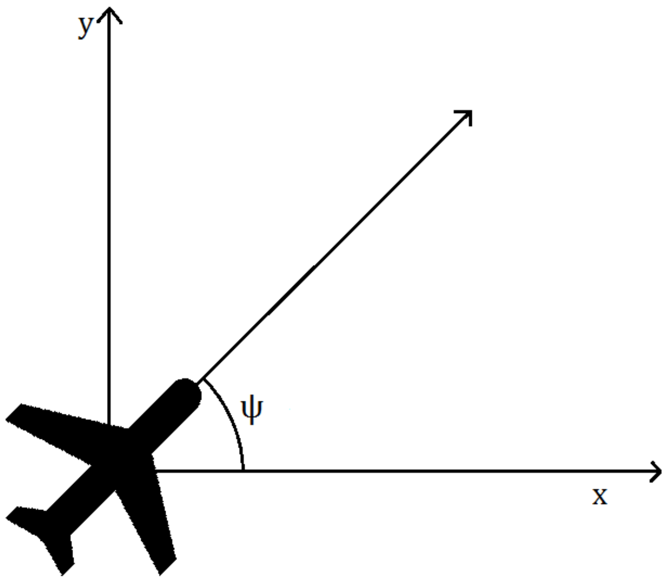

Figure 2-3: Dubins vehicle model, x is east position, y north and ψ is the heading.

A mathematical model for a Dubins vehicle with constant speed U and turn-rate input u is:

$$\begin{aligned} \dot{x} &= U\cos(\psi) \\ \dot{y} &= U\sin(\psi) \\ \dot{\psi} &= u \end{aligned} \tag{2.1}$$

$$u_{min} \leq u \leq u_{max}$$

2.2 Nonlinear Optimization

In mathematics, optimization is the minimization or maximization of a function subject to constraints on its variables [15]. The function is called an objective function and measure the performance of a certain solution. It consists of parameters such as profit, time, cost, energy or a combination of parameters, depending on the problem, as long as it is a scalar function, meaning that it can be represented by a number. Together, the objective function, the variables and the constraints are called an optimization problem.

2.2.1 Formulation

An optimization problem is often formulated as follows:

$$\min_{x\,\epsilon\, R^n} f(x) \quad subject\ to \quad \begin{matrix} c_i(x) = 0 & i\,\epsilon\,\mathcal{E} \\ c_i(x) \geq 0 & i\,\epsilon\, I \end{matrix} \tag{2.2}$$

In equation (2.2), the objective function $f(x)$ is minimized subject to the constraints c_i, often divided into equality constrains $i\,\epsilon\,\mathcal{E}$ and the inequality constraints $i\,\epsilon\, I$.

Once the problem has been formulated, there exist a number of different methods of solving the specific problem. These have various pros and cons and it falls to the user to choose a fitting method for the problem at hand. The method used in this project is an interior point method.

2.2.2 The Lagrangian function

The Lagrangian function is central to nonlinear optimization. It introduces a new variable λ, called a Lagrange multiplier. The Lagrange function is defined as:

$$\boldsymbol{\mathcal{L}}(x, \lambda_i) = f(x) - \lambda_i c_i(x) \tag{2.3}$$

The gradient of the Lagrange function is:

$$\nabla_x \boldsymbol{\mathcal{L}}(x, \lambda_i) = \nabla f(x) - \lambda_i \nabla c_i(x) \tag{2.4}$$

Combined with the fact that at a solution x^* the constraint normal $\nabla c_i(x^*)$ is parallel to $\nabla f(x)$, there exists a multiplier λ_i^* such that:

$$\nabla f(x^*) = \lambda_i^* \nabla c_i(x^*) \tag{2.5}$$

An equality constraint can then be found and used later in the Karush-Kuhn-Tucker (KKT) conditions.

$$\nabla_x \boldsymbol{\mathcal{L}}(x^*, \lambda_i^*) = 0 \tag{2.6}$$

2.2.3 The Karush-Kuhn-Tucker conditions

In optimization theory, the Karush-Kuhn-Tucker (KKT) conditions are the first order necessary conditions for a solution to be optimal. Given a local solution x^* of the problem, where the linear independence constraint qualifications (LICQ) are valid, and that the functions f and c_i are continuously differentiable, there exists a Lagrange multiplier vector λ^* , with components $\lambda_i^*, i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions are satisfied at (x^*, λ^*).

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0 \\ c_i(x^*) &= 0 \quad for\ all\ \ i \in \mathcal{E} \\ c_i(x^*) &\geq 0 \quad for\ all\ i \in \mathcal{I} \\ \lambda_i^* \ &\geq 0 \quad for\ all\ i \in \mathcal{I} \\ \lambda_i^* c_i(x^*) &= 0 \quad for\ all\ \ i \in \mathcal{E} \cup \mathcal{I} \end{aligned} \tag{2.7}$$

2.2.4 Interior point method

The method used to solve the optimization problems in this project are of the type known as interior point or barrier methods. The interior point method works by traversing the interior of the feasible region for the problem. This method is known for its fast calculations on large systems with sparse structure, meaning that most of the elements in the matrix are zero.

Given a general nonlinear optimization problem with s as a slack variable:

$$\min_x f(x) \tag{2.8}$$

$$subject\ to\!: \begin{aligned} c_e(x) &= 0 \\ c_i(x) - s &= 0 \\ s\ &\geq 0 \end{aligned}$$

An interior point algorithm finds a solution by solving a series of barrier problems [15]. These problems are quite similar to the regular optimization problem, but introduces a barrier term $-\mu \Sigma_{i=0}^{\mathrm{n}} \log s_i$, which prevents the components of s to becoming too close to zero, so the inequality in (2.8) can be removed.

$$\min_x f(x) - \mu \sum_{i=1}^{n} \log s_i \tag{2.9}$$

$$subject\ to\!: \begin{aligned} c_e(x) &= 0 \\ c_i(x) - s &= 0 \end{aligned}$$

Then a barrier function $P(x;\mu)$ is introduced.

$$P(x;\mu) = f(x) - \mu \sum_{i \,\epsilon\, \mathcal{I}} \log s_i \tag{2.10}$$

This function will keep x inside the feasible region since the logarithmic term will increase drastically if it comes close to the boundaries. Problems can occur if the solution is close to the edge of the feasible region, but most modern algorithms have methods of dealing with this. Over the series of iterations of the barrier problems, the parameter μ is decreased, and when it approaches zero the minimal solution to the barrier problem will be equal to the original problem (2.8).

2.3 Optimal Control

Optimal control is a way to use optimization algorithm for control purposes. As with an optimization problem, the control problem includes an objective function which here consists of state- and control variables. A solution to an optimal control problem is usually a set of inputs to a system which minimizes the objective function.

2.3.1 General formulation

In this project, the solution for the optimal control problem will be a vector consisting of control inputs to the UAV which will make the UAV fly through the given waypoints. A general way to denote the optimal control problem (OPC) is as follows:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \boldsymbol{J}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{t})\,; \quad \boldsymbol{J} = \boldsymbol{\Phi}\left[\, x(t_o), t_0, x\left(t_f\right), t_f\right] + \int_{t_0}^{t_f} \boldsymbol{\mathcal{L}}\,[x(t), u(t), t]dt$$

$$\begin{aligned} &subject\ to: \\ &\dot{x}(t) = \boldsymbol{a}[x(t), u(t), t] \\ &\boldsymbol{b}[x(t), u(t), t] \leq 0 \\ &\boldsymbol{\phi}\left[x(t_0), t_0, x\left(t_f\right), t_f\right] = 0 \end{aligned} \tag{2.11}$$

In (2.11), $\boldsymbol{J}$ is the cost-function (also called objective function), $x(t)$ and $u(t)$ is the state and control input respectively. t is the continuous time, t_0 is the initial time and t_f is the terminal time, often called end time. The term $\boldsymbol{\Phi}$ is called the end point cost or the Mayer term, $\boldsymbol{\mathcal{L}}$ is called the continuous cost or the Lagrange term. $\boldsymbol{a}$ is the system's state equation, $\boldsymbol{b}$ is the inequality constraints and $\boldsymbol{\phi}$ is the equality constraints.

2.3.2 Direct collocation

Direct collocation is the most common of the direct transcription methods. In this optimal control method, both the state and the controls for the optimal control problem are discretized. The time horizon is split into a rather fine grid of N so-called collocation intervals $\{t_k\}_{k=0}^{N}$. On this grid we denote the state variables $s_k \approx x(t_k)$ and a control parameter q_k such that $u_k(t; q_k)$. Then, on each of these collocation intervals $[t_k, t_{k+1}]$ a set of d collocation points are chosen, often as either Gauss-Legendre or Gauss-Radau point as shown in Table 2-1.

Table 2-1: Gauss-Legendre and Gauss-Radau points of orders 1-4

Order	**Gauss-Legendre points**	**Gauss-Radau points**
1	0.5	1.0
2	0.211325, 0.788675	0.333333, 1.0
3	0.112702, 0.5, 0.887298	0.155051, 0.644949, 1.0
4	0.069432, 0.330009, 0.669991, 0.930568	0.088588, 0.409467, 0.787659, 1.0

The constraints for the optimal control problem is based on the collocation conditions which are:

1. The initial condition: $s_k = p(t_k; v)$
2. The differential condition: $f\left(p\left(t_k^i; v\right), u_k\left(t_k^i; q_k\right)\right) = p'\left(t_k^i; v\right), \; i = 1 \ldots d$

In addition, it is required that there is continuity between the collocation intervals, meaning that the end of one interval is the start of the next one; $p_k(t_{k+1}; v) = s_{k+1}$. The resulting optimal control problem becomes [16]:

$$\min_{s,v,q} \quad \sum_{k=0}^{N-1} l_k(s_k, v_k, q_k) + E(s_n) \tag{2.12}$$

$$\begin{array}{rll}
\mathit{Subject\ to}: & & \\
s_0 - x_0 = 0 & & \mathit{fixed\ initial\ value} \\
c_k(s_k, v_k, q_k) = 0 & k = 0, \dots, N-1 & \mathit{collocation\ conditions} \\
p_k(t_{k+1}; v_k) - s_{k+1} = 0 & k = 0, \dots, N-1 & \mathit{continuety\ conditions} \\
h(s_k, q_k) \leq 0 & k = 0, \dots, N-1 & \mathit{path\ constraints} \\
r(s_N) \leq 0 & & \mathit{termal\ constraints}
\end{array}$$

2.4 MODEL PREDICTIVE CONTROL

2.4.1 MPC overview

Model predictive control (MPC) is an advanced control method which builds upon optimal control. It is mainly used in process industries such as chemical plants, oil refineries and power plants. Model predictive controllers use an optimization algorithm based on a model of the plant to find the inputs the system needs to follow the optimal trajectory. This optimization is done recursive, always with the most recent system state as the initial condition. This means that only the first couple of inputs from the optimizations are used, since when a new optimization is ready, the inputs from the most recent are used instead. The optimizations are done on a finite time-horizon, to reduce computational cost. One of the greatest advantages to MPC is the ability to anticipate future events and take action according to these. In MPC theory one usually divides the system into three different types of variables; the control inputs, usually denoted as u, are called the manipulated variables (MV), the system states are called the controlled variables (CV), and disturbance variables (DV).

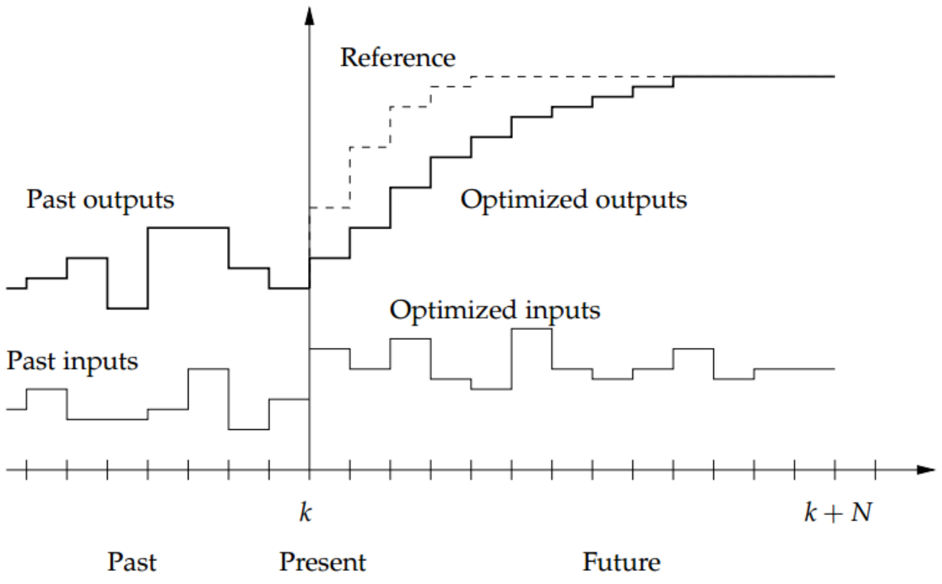

Figure 2-4: MPC principle.

2.4.2 MPC formulation

Since the MPC is solving an optimal control problem for each iteration, so the formulation of the controller problem is identical to the optimal control problem using collocation (2.12):

$$\min_{s,v,q} \qquad \sum_{k=0}^{N-1} l_k(s_k, v_k, q_k) + E(s_n) \tag{2.13}$$

$$\begin{array}{rll}
\textit{Subject to}: & & \\
s_0 - x_0 = 0 & & \textit{fixed initial value} \\
c_k(s_k, v_k, q_k) = 0 & k = 0, \dots, N-1 & \textit{collocation conditions} \\
p_k(t_{k+1}; v_k) - s_{k+1} = 0 & k = 0, \dots, N-1 & \textit{continuety conditions} \\
h(s_k, q_k) \le 0 & k = 0, \dots, N-1 & \textit{path constraints} \\
r(s_N) \le 0 & & \textit{termal constraints}
\end{array}$$

2.4.3 MPC design

When designing a model predictive controller, there are a few important design decisions to be made. The choice of optimization model is one of these. There is a trade-off between a simple model for fast optimization, and thus more regular updates, and a detailed model for more accurate results, but slower optimizations and fewer iterations. A general “rule-of-thumb” when choosing an optimization model is that it is must be able to capture the most significant dynamics of the system. The remaining un-modelled dynamics of the system are often called plant-model mismatch.

While the idea behind is the same, MPC controllers can be implemented in a number of different ways, depending on where in the so-called control hierarchy they are placed. Usually the MPC controller does not control the system directly, but is used to calculate set-points for local controllers, such as PID-controllers. A standard MPC hierarchy is shown in Figure 2-5.

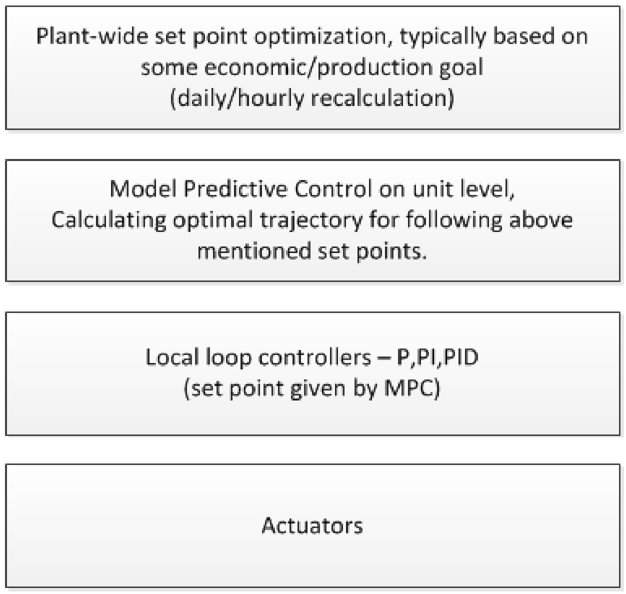
Plant-wide set point optimization, typically based on some economic/production goal (daily/hourly recalculation)
Model Predictive Control on unit level, Calculating optimal trajectory for following above mentioned set points.
Local loop controllers – P,PI,PID (set point given by MPC)
Actuators

Figure 2-5: Standard MPC hierarchy.

When using model predictive controllers for a fast system such as an UAV, MPC is in most cases used for guidance purposes instead of directly controlling the vehicle. This is because of the delay in the MPC-loop and deviations due to model mismatch, which without low-level regulators or set-point control can make the system deviate from the optimal solution. However, in this project the MPC inputs was used directly. The turn-rate acquired from the optimizations were sent directly to the UAV plant model. In this setup the MPC worked as both a guidance and control system for the vehicle, in the sense that it both found the optimal path and was used to directly control the actuators on the UAV.

2.5 BISECTION METHOD

Bisection, or binary search, is a root-finding method in which an interval is split into two sections. These sections are then evaluated to find the section containing the solution, before splitting the correct section again. This can, after a few iterations, give a good approximation of where the solution can be found. In Figure 2-6, a bisection on a function $f(x)$ is shown. The first interval is found by simply guessing the first two x values and checking the function at these. If they have opposite signs, the root is known to be in between, and the middle value of these guesses are then checked.

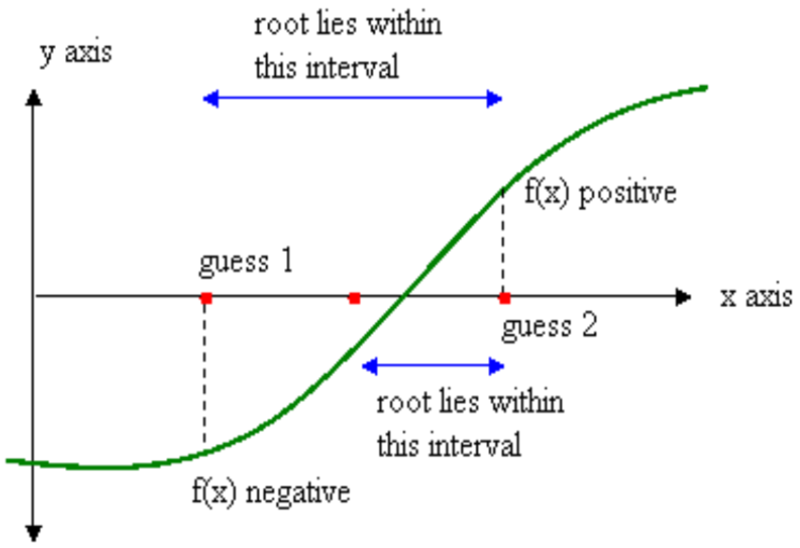

Figure 2-6: Bisection root-finding method, narrowing down the interval where the root lies.

In this project, a bisection algorithm was used in the optimal controller to find a good end time for the optimization. This will be more thoroughly explained in the Implementation chapter.

2.6 SOFTWARE

2.6.1 Python

Python is a high level programming language, which is designed with emphasis on readability and simplicity. It supports objective oriented programming and has a large standard library. In contrast to languages like C and C++, python code is interpreted instead of compiled, which means that the code is run by an interpreter, instead of being translated into machine code, and run directly on the system. With additional open source packages such as NumPy, SciPy and Casadi, the Python library is vastly expanded and becomes a complete tool for scientific computations, similar to premium programs like MATLAB.

2.6.2 Casadi

Casadi [17] is a symbolic framework for C++ and Python. This framework makes it possible for the user to do differentiation and numeric optimization using algebraic syntax. The framework can be used with both scalar and matrix variables. Casadi uses modern algorithms for the algorithmic differentiation, while keeping it relatively simple to use. The use of Casadi in this project is motivated by the fact that Casadi is very efficient to calculate derivatives [18], which makes for fast computational times for the optimal controller.

2.6.3 IPOPT

The interior point optimizer [19] (IPOPT) is an open source software package for large-scale nonlinear optimization. The algorithm used in IPOPT is the Primal-Dual Interior Point algorithm.

2.6.4 MA27 solver

To reduce the frequency of unfeasible solutions during the recovery phase of the IPOPT algorithm, the MA27 [20] solver was acquired and used.

3 IMPLEMENTATION

The implementation was done in the programming language Python, with heavy use of the Casadi [17] symbolic toolbox. This toolbox is efficient in nonlinear numerical optimization, and supports most of the state-of-the-art codebases such as Sundials, WORHP and IPOPT [19] which is used in this project. Most of the Casadi code is written in C++ and is just interfaced with use of Python. The specific solver used was the HSL MA27 [21] solver. The plant model was simulated between the optimizations using a standard Runge-Kutta integrator.

3.1 DEFINING THE TEST SCENARIO

3.1.1 The icebergs

The test scenario consists of six icebergs spread out in a pattern. The order in which the icebergs is to be visited is predefined. This order could be found solving a version of the traveling salesman problem [22], as was done in [23].

Table 3-1: The icebergs for the test scenarios

#Iceberg	**0 (init)**	**1**	**2**	**3**	**4**	**5**	**6**
Position (x,y)	(0, 0)	(0, 750)	(1000, 600)	(1300, 1500)	(350, 1600)	(500, 2500)	(-1000, 2500)

3.1.2 UAV model

For the UAV a kinematic model was used. This means that it just describes the motion the UAV and not what causes the motion. It was based on a Dubins vehicle with constant speed and limited turn-rate. This is quite realistic in the sense that a plane-like UAV cannot stop, since it will fall down and crash, and it has to bank in order to turn. In addition, only two dimensions are of interest in this case of waypoints following, thus the altitude dimension was removed from the model, and only x- and y-positions and the yaw-angle were left.

$$\begin{aligned}\dot{x} &= U\,cos(\psi) \\ \dot{y} &= U\sin(\psi) \\ \dot{\psi} &= u\end{aligned} \tag{3.1}$$

In the Dubins vehicle model for the UAV, shown in equation (3.1), U is the constant speed, x is the east position, y is the north position and ψ is the yaw-angle. The UAV model was implemented using Casadi using the code in Code 3-1.

```
#Optimization model
x = SX.sym('x', 3)    #states x, y, psi
u = SX.sym('u')       #control u

#ODE right hand side
x_dot = ([ U*cos(x[2])+W[0]*cos(W[1]),
           U*sin(x[2])+W[0]*sin(W[1]),
           u ] )
f_opt = SXFunction([ x, u], [x_dot])
f.opt.init()
```

Code 3-1 Initializing the optimization model, using symbolic Casadi variables.

3.1.3 Wind disturbance

Winds in the arctic can reach up to 50 m/s in storms, however the average wind is around 5 m/s in summer and 9 m/s in winter [20]. The small size of the UAV in this scenario, makes it not viable to fly during very strong winds, thus the storm strength winds were not included in this project. But both the winter average and summer average wind speeds were tested.

Since the UAV model is a kinematic model, the wind disturbance can be added as a separate term that contributes to the position change for the UAV. The wind terms was modelled with a magnitude W and direction ω. Other wind models often include a mean strength and a random gust applied on top of the mean value. In this project, to keep things simple, only the mean value was included, as seen in equation (3.3). However both the mean magnitude and direction values were set to change over time.

$$\begin{aligned} \dot{x} &= W\,cos(\omega) \\ \dot{y} &= W\sin(\omega) \end{aligned} \tag{3.2}$$

The wind is thought to only contribute to the position of the UAV, and thus it is not present in the equation for the heading angle. When the wind term is added into the equation (3.1) of the UAV, the UAV dynamics and the wind disturbance together becomes:

$$\begin{aligned} \dot{x} &= U\,cos(\psi) + W\,cos(\omega) \\ \dot{y} &= U\,\sin(\psi) + W\sin(\omega) \\ \dot{\psi} &= u \end{aligned} \tag{3.3}$$

3.2 Optimal Control Implementation

3.2.1 Overall optimization design

The optimal controller was designed to fly the UAV over a series of icebergs, represented here by a set of waypoints. It was decided to split the optimization problem into smaller parts, where each part is the path from one waypoint to the next. Using this approach, it was possible to keep the optimization simple, since it avoids having to do event detection for the icebergs during the optimization itself. Another benefit is that the MPC controller does not have to optimize the whole time horizon in each iteration. All these smaller paths were then added together to form the final solution for the problem.

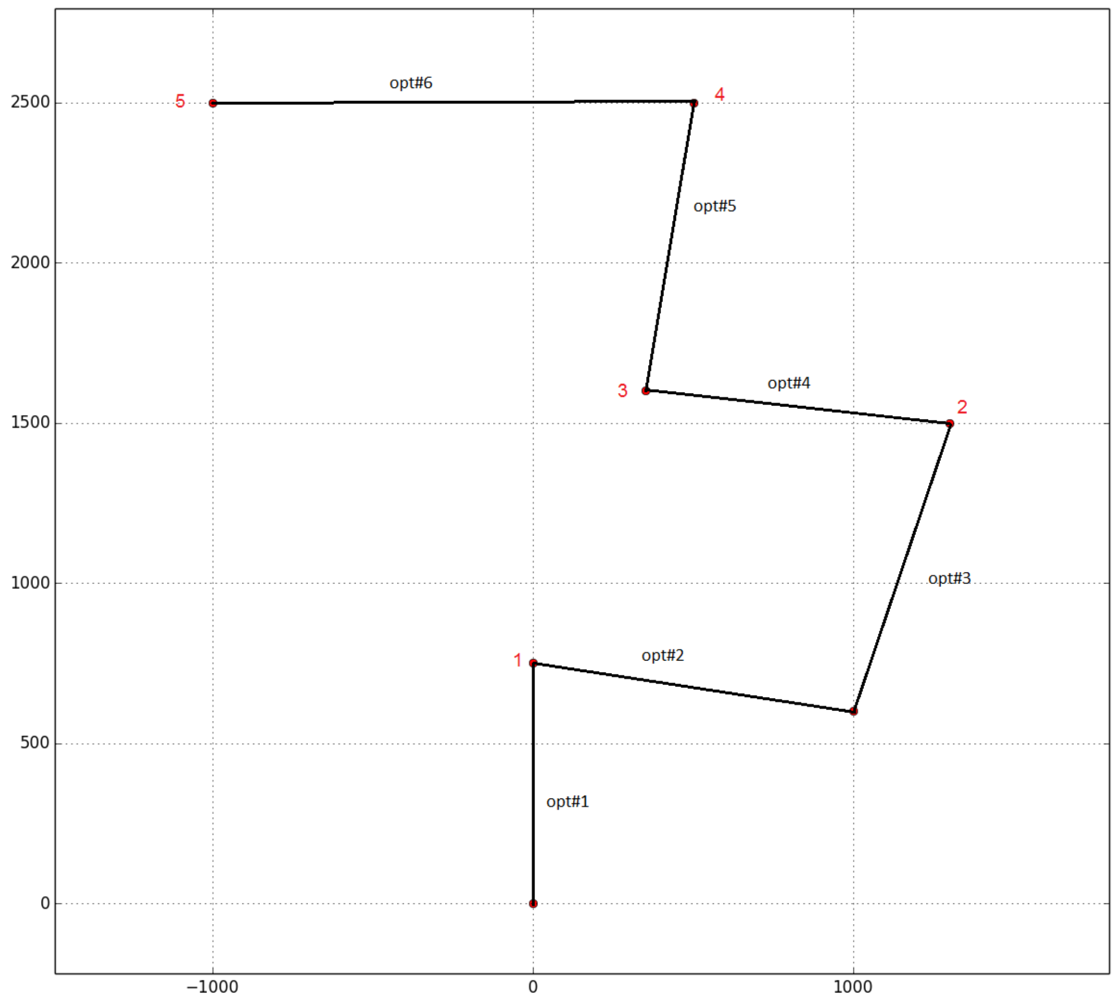

Figure 3-1: Splitting the large problem into multiple smaller problem.

One challenge with this design is that both the start and end-positions are fixed, and the UAV model does not allow changes in speed. This makes the time horizon the key parameter for the solution, since the UAV must be at the next waypoint at the end of this horizon. An optimization scheme with the time horizon as part of the objective function was tried, but the added complexity to the problem made it very hard to find a feasible solution. The scheme that eventually was implemented, was one where the time horizon is first approximated with geometry, then a bisection search was done around this estimation to find an optimal time horizon. In pseudocode the optimization was implemented as follows Code 3-2:

```
Calculate collocation matrices
Calculate the desired target heading angle at each waypoint
For each waypoint:
    Formulate the current constraints and objective function
    Calculate an approximate time to reach the waypoint
    While not satisfied by the solution:
        Initialize and run the optimization
        Improve the approximated time based on the result
```

Code 3-2: Pseudocode of the optimal control problem.

3.2.2 Fixed end position scheme

The fixed end position scheme was used to minimize the trajectory by making the UAV to be at a certain waypoint and with a set heading angle at the end of the time-horizon. The target angle was taken from the geometry behind Dubins path generation, where the optimal path intersects the waypoints in the middle of the required turn. This means that the heading angle for each iceberg can be calculated by simple geometry, prior to the initialization of the optimization problem. Since the larger problem was split into smaller parts, each sub-problem used the solution from the last optimization as its initial position. This was also the case when the MPC-functionality was added, where the initial position for the optimization was where the UAV was at the end of the previous optimization. This keeps the continuity between the solutions of the sub-problems and the resulting optimal path, when combining the optimal sub-paths, should be the optimal solution to the larger problem.

3.2.3 Direct collocation implementation

To solve the optimal control problem, the direct collocation method was used. As described in chapter two, the direct collocation method discretizes both the controls and the states of the system. The resulting NLP problem when using the collocation method was:

$$\min_{s,v,q} \qquad \sum_{k=0}^{N-1} l_k(s_k, v_k, q_k) + E(s_n) \tag{3.4}$$

$$\begin{array}{rll}
Subject\ to: & & \\
s_0 - x_0 = 0 & & fixed\ initial\ value \\
c_k(s_k, v_k, q_k) = 0 & k = 0, \dots, N-1 & collocation\ conditions \\
p_k(t_{k+1}; v_k) - s_{k+1} = 0 & k = 0, \dots, N-1 & continuety\ conditions \\
h(s_k, q_k) \leq 0 & k = 0, \dots, N-1 & path\ constraints \\
r(s_N) \leq 0 & & termal\ constraints
\end{array}$$

The constraints in this problem were calculated with two coefficient matrices $\boldsymbol{C}$ and $\boldsymbol{D}$, for the collocation equation and the continuity equations respectively. Firstly, a Lagrange polynomial of the collocation points using Gauss-Legendre points of order *3* was made. A higher order of Legendre points would give a more accurate discretization of the system, but with a cost of longer calculations. The basis of the Lagrange polynomial is shown in equation (3.5).

$$\ell_j(\tau) = \prod_{r=0 \neq j}^{d+1} \frac{\tau - \tau_{d_r}}{\tau_{d_j} - \tau_{d_r}} \tag{3.5}$$

$$\tau_d = (0,\ 0.112702, 0.5, 0.887298)$$

To calculate the $\boldsymbol{D}$ matrix for the continuity equation the Lagrange basis polynomial is evaluated at the end of the collocation interval for each of the polynomials, which is $\tau = 1$. Together the $\boldsymbol{D_j}$ parts form the matrix $\boldsymbol{D}$.

$$\boldsymbol{D}_j = \ell_j(\tau = 1) \tag{3.6}$$

$$\boldsymbol{D} = [\, D_0, \dots, D_{d+1}]$$

However, to make the $\boldsymbol{C}$ matrix, that is the collocation equation, the derivative of the Lagrange basis polynomial is needed. This derivative was then evaluated at each of the collocation points to get the matrix $\boldsymbol{C}$.

$$\boldsymbol{C}_{j,r} = \frac{d\ell_j}{d\tau}\,(\tau_{d_r}) \tag{3.7}$$

$$\boldsymbol{C} = \begin{bmatrix} C_{j=0,r=0} & \cdots & C_{j=0,r=d+1} \\ \vdots & \ddots & \vdots \\ C_{j=d+1,r=0} & \cdots & C_{j=d+1,r=d+1} \end{bmatrix}$$

With the $\boldsymbol{C}$ and $\boldsymbol{D}$ matrices completed, they can be used to formulate the continuity constraints for the optimization problem.

3.2.4 The optimal control problem formulation

As mentioned earlier, the optimization problem was split into multiple parts, each consisting of the path from one waypoint to the next. On each of these smaller optimizations, the start and end positions for the UAV was fixed. To increase the chance for a feasible solution, the objective was given slack in both position and heading at the end point. This slack was implemented by adding an accuracy term in the constraints on the end position as shown in Code 3-3

```
#Sate bounds
x_min = [-inf, -inf, -inf] #Minimun bounds for the UAV
x_max = [ inf, inf, inf]   #Maximum bounds for the UAV
xi_min = position     #Initial min bound for the UAV
xi_max = position     #Initial max bound for the UAV
xf_min = x_target[i] - x_accuracy     #End position min bounds
xf_max = x_target[i] + x_accuracy     #End position max bounds
```

Code 3-3: Constraints for the initial and end position of the UAV.

As mentioned in chapter two, an optimal control problem consists of the objective function, the system variables, the constraints based on the dynamic model, and the other constraints set by the user.

Since the problem was so heavily constrained, the chosen objective function kept simple. Its main purpose was to prioritize solutions as close to the centre of the error margin as possible. It consists of two parts; the end-cost or Mayer term, to minimize the distance to the waypoint at the end position, and the Lagrange term, to minimize the amount of turning the UAV has to do over the path. The resulting objective function was:

$$J(x,u) = \sum_{t_0}^{t_f} qu(t)^2 + \left\| x_{wp} - x(t_f) \right\| \tag{3.8}$$

The variables for the control problem consist of the collocation states and the control inputs over the control interval. At each of the n control intervals, there are $(d+1)$ collocation states, each with n_x variables, and one control input u. The total number of states then become: $w = n(\, n_x(d+1) + u)$. The variable vector w was then initialized and split into a part X, with all the states, and a part U, with all the inputs. Constraints, such as the limited turn-rate were generated and stored in vectors to initialize the system with, the complete code to generate all the states and their respectively constraints are shown in Code A-1 in the appendix.

The collocation- and continuity constraints were generated using the collocation matrices $\boldsymbol{C}$ and $\boldsymbol{D}$ found earlier. The $\boldsymbol{C}$ matrix, which consists of derivatives of the Lagrangian function was used to make the constraints that make up the differential collocation condition; $f\left(p(t_k^i; v), u_k(t_k^i; q_k)\right) = p'(t_k^i; v)$. The $\boldsymbol{D}$ matrix, consisting of the Lagrangian polynomial evaluated at the end of each collocation interval was used to make the constraints that result in the continuity conditions. The code is shown in Code 3-4 below.

```
for k in range(n): #For all finite elements
     for j in range(1,d+1): #For all collocation points
          xp_jk = 0 #state derivative at the collocation point
          for r in range (d+1):
               xp_jk += C[r,j]*X[k,r]
          #Add collocation equations to the NLP
          [fk] = f_opt.call( [ X[k,j], U[k] ] )
          g.append(DT*fk - xp_jk )
     #Get an expression for the state at the end
     xf_k = 0
     for r in range(d+1):
          xf_k += D[r]*X[k,r]
     #Add continuity equation to NLP
     g.append(X[k+1,0] - xf_k)
```

Code 3-4: Generating the collocation constraints for the NLP

With all the parts of the optimal control problem complete, these were assembled together to the final optimization problem. Code 3-5 shows the initialization of the optimization.

```
#NLP
nlp = MXFunction(nlpIn(x=w), nlpOut(f=Obj, g=g) )
solver = NlpSolver("ipopt", nlp)
solver.setOption("expand", True)
solver.setOption("linear_solver", "ma27")
solver.init()
solver.setInput(vars_init, 'x0')
solver.setInput(vars_lb, 'lbx')
solver.setInput(vars_ub, 'ubx')
solver.setInput(NP.concatenate(lbg), 'lbg')
solver.setInput(NP.concatenate(ubg), 'ubg')
solver.evaluate()
```

Code 3-5: Initializing the NLP and starting the optimization.

The MA27 [21] solver from HSL was acquired with an academic licence form the HSL team to fix a problem during the restoration phase of the IPOPT algorithm. Implementation of the time Bisection

As pointed out earlier, the time horizon itself is a key factor in the optimal solution. Therefore, a bisection method was implemented to find an optimal end time. First, an approximation of the end time, t_f, is calculated. This approximation uses the distance to the waypoint and the total curvature of the path the UAV has to follow. In the scenarios with wind disturbance, the wind component parallel to the path between the waypoints, was also taken into account. This was done by altering the speed for the UAV in that direction when calculating the approximation.

$$t_f = \frac{distance}{UAV_{Speed} + W_{W||wp}} + k_\psi |\psi_{target} - \psi_{init}| + t_{bisection} \tag{3.9}$$

Starting with the approximated time t_f, the time is then either increased or decreased depending on the result from the optimization algorithm.

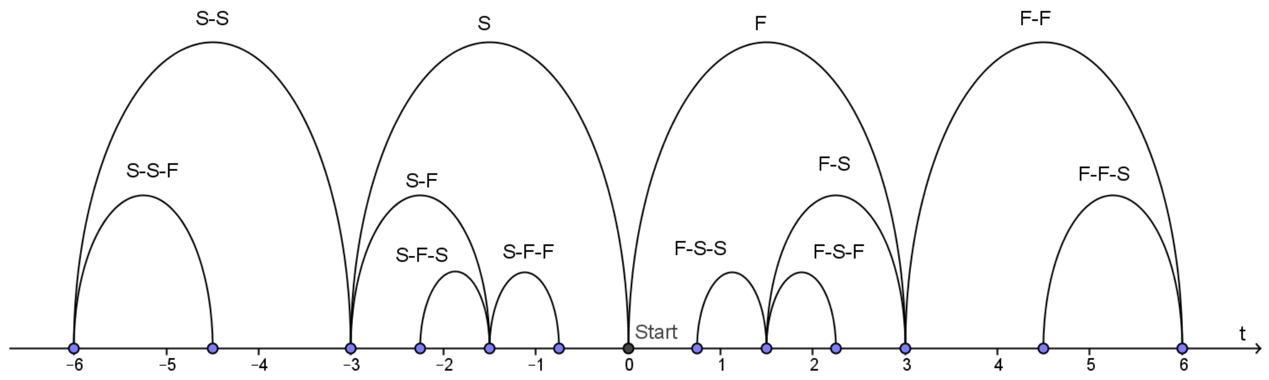

Figure 3-2. Bisection on the end time, each S means a success and each F means a failure. Thus, S-F is the sequence success - failure.

In Figure 3-2 the bisection method is shown. In the figure, an **S** means a successful optimization, and **F** a failure. **S-F** means that the first optimization was a success and the second a failure, and so on. Suitable values for the added and subtracted times for the bisection algorithm were found after tuning. The code for the bisection logic is found in Code A-2 in the appendix.

3.2.5 Compensating for wind disturbance

The optimization model was expanded to include an estimate of the wind conditions, $\widehat{W}$ for wind strength and $\widehat{\omega}$ for direction. These are set at the start of the each optimization, and are viewed as constant for the duration of each optimization. Otherwise, they are implemented in the same way as the wind in the plant model.

$$\begin{aligned} \dot{x} &= U\, cos(\psi) + \widehat{W}\, cos(\widehat{\omega}) \\ \dot{y} &= U\, \sin(\psi) + \widehat{W}\, \sin(\widehat{\omega}) \\ \dot{\psi} &= u \end{aligned} \tag{3.10}$$

Since the wind is viewed as constant during the optimization horizon, the additional terms do not add more states or variables to the system, meaning that they do not increase the computational time for the optimizations. The additional terms do, however, reduce the feasible region so that an optimal solution is harder to find.

3.3 MODEL PREDICTIVE CONTROL IMPLEMENTATION

3.3.1 MPC structure

As described in chapter two, model predictive controllers can vary in the way they are constructed. In this project, the MPC is used as both a guidance and control system. This means that the inputs are taken directly from the optimizations and used on the UAV model. An overview of the system is shown in a block diagram in Figure 3-3. Here, the controller is given the waypoints, position measurements of the UAV, and an estimation of the wind. In most of the simulations the wind information is given at the start of the simulation and held constant over the course of the simulation, while the actual wind disturbance is allowed to change.

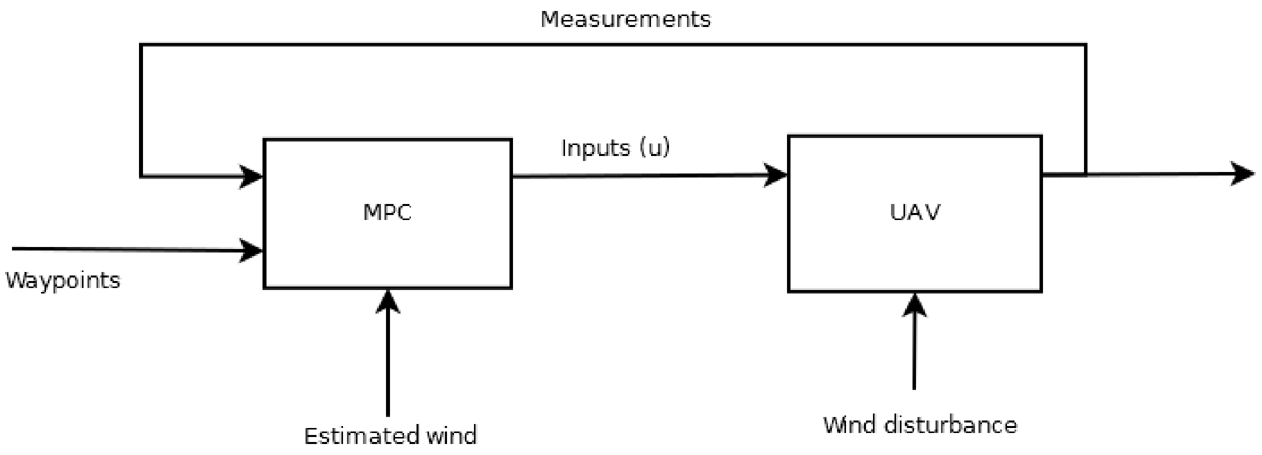

Figure 3-3: MPC design overview for the system.

The optimization horizon for the controller is a measure of how far into the future the controller optimizes. Since only the first couple of inputs of each optimization are used, the controller does not need the inputs for the whole path through all the waypoints. The optimization horizon for the controller was set to be the next three waypoints. This decreases the computational time for each of the iterations of the MPC algorithm. The MPC algorithm is explained in pseudocode in Code 3-6.

```
Generate targets angles from waypoints
While not visited all waypoints:
    Select the three next waypoints
    Run optimization to find the optimal path from current position
    If successful:
        Use most recent control inputs on the plant model
    If infeasible:
        Use previous successful control inputs on the plant model
    Update wind disturbance
    Update position
    Remove visited waypoints from waypoints list.
```

Code 3-6: Pseudocode for the MPC algorithm.

3.3.2 **Plant model**

A plant model was implemented to act as a substitute for the real UAV. This model takes the control inputs from the MPC algorithm and simulates how a UAV would fly with these inputs. The plant model, as the optimization model, is based on a Dubins vehicle model with the added wind terms, as in (3.3). In addition, a limitation is placed on the change in turn-rate, so the UAV cannot go instantly from full turn (bank) in one direction to full turn in the opposite direction.

$$\begin{aligned}\dot{x} &= U\, cos(\psi) + W(t)\cos(\omega(t)) \\ \dot{y} &= U\ \sin(\psi) + W(t)\sin\left(\omega(t)\right) \\ \dot{\psi} &= u\end{aligned} \tag{3.11}$$

$$|\ddot{\psi}| \leq \delta_{tr}$$

In order to simulate the UAV, a regular Runge-Kutta integrator was used. Since this is a discrete time integrator, the system must first be discretized. The equations of the discrete system become:

$$\begin{aligned} x_{k+1} &= x_k + h(\ U\,cos(\psi) + W cos(\omega)\ ) \\ y_{k+1} &= y_k + h(\ U\,sin(\psi) + W \sin(\omega)\ ) \\ \psi_{k+1} &= \psi_k + h\,u \end{aligned} \tag{3.12}$$

$$|\psi_{k+1}| \leq \delta_{tr}\ h\ \psi_k$$

The code for the Runge-Kutta integrator used is shown in Code A-3 in the appendix.

3.3.3 Handling infeasible solutions

How well a model predictive controller handles infeasible solutions is an important factor in how safe and reliable it is. This problem can be dealt with in a number of different ways. One way is to have a backup controller which can take control if no solution is found. This controller could use a simple waypoint following algorithm, such as LOS. However, since there already is calculated an optimal path, a path following algorithm using the already generated path would be preferable.

Since the main focus of this project is on the main controller loop, the method chosen in this project was to simply use the inputs from the previous solution until a new feasible solution is found. While not perfect, this method works in most cases.

3.3.4 Simulation time

The simulation time for each iteration of the model predictive controller was first set to be the same length as the computational time used by the optimizations. The optimizations are slower in the cases where they do not find a feasible solution, due to how the optimization algorithm works. This slowdown can sometimes also be amplified by the bisection method, since a series of optimizations are run. This increase in computational time causes the system to wait longer for updated control inputs. The delays may cause small course deviations to become a larger problem, since the needed inputs to correct the heading might come too late. A fixed simulation time was introduced, and as expected, made the controller perform better near the waypoints, where most of the optimization failures occurred. The fixed simulation time is not as realistic as the dynamic simulation time. However, the computational time is scalable with more powerful computer hardware. Thus, finding a simulation time where the update frequency of the inputs is no longer a problem, gives an idea of how powerful the hardware must be to run the model predictive controller. The fixed simulation time was, after some testing, set to 5 seconds.

4 SIMULATIONS AND RESULTS

To test the performance for the whole system, a series of simulations were run. In these simulations, the complexity of the controller and the difficulty of test scenarios were incrementally increased to test the robustness of the controller. The first section of this chapter concerns just the optimal controller and the last section concerns the performance of the whole model predictive controller.

4.1 OPTIMAL CONTROLLER PERFORMANCE

First in this section, the path generated by the optimal controller is shown, and thereafter, the effect of the bisection algorithm. Lastly, the path and heading angle from an optimization which compensates for a wind disturbance, is compared to an undisturbed solution.

4.1.1 Optimization algorithm

In Figure 4-1, a path generated by the optimal controller is shown. This optimization was done with no wind disturbance, and shows an ideal scenario. As seen in the figure, the UAV intersects the waypoints in the middle of each turn. This is the behavior is identical to the Dubins path, and makes the shortest path available through the waypoints in the order they are given.

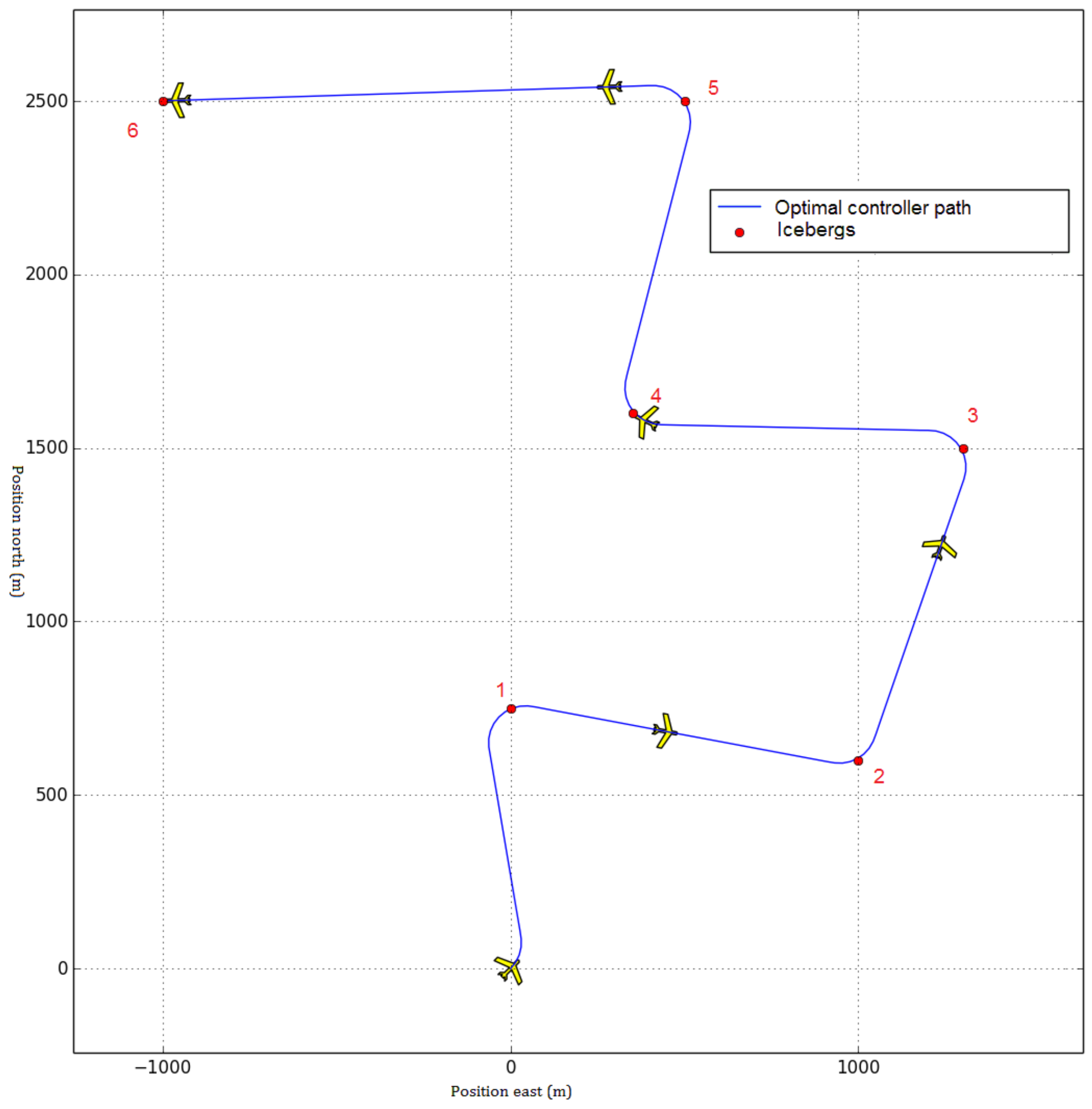

Figure 4-1: Ideal path generated by optimization algorithm, with no wind disturbance.

4.1.2 End time bisection effect

As mentioned earlier, the bisection algorithm works by adding or subtracting time to/from the time horizon for the optimal controller. The bisection time in Table 4-1 is the $t_{biseciton}$ from equation (3.9). This table and Figure 4-2 show that the first two optimizations were failures, and that the optimization needed six seconds additional simulation time to find a feasible solution. After the first success the next optimization uses the mean of the last success and last failure, in this case 4.5

seconds. This optimization is also a success and a new simulation is done with 3.75 seconds additional time, which is the last optimization performed on this section.

Table 4-1: Bisection of the end-time

Simulation:	**1**	**2**	**3**	**4**	**5**
Result:	Infeasible	Infeasible	Success	Success	Success
Bisection time (t_{bisec}):	0s	3s	6s	4.5s	3.75s

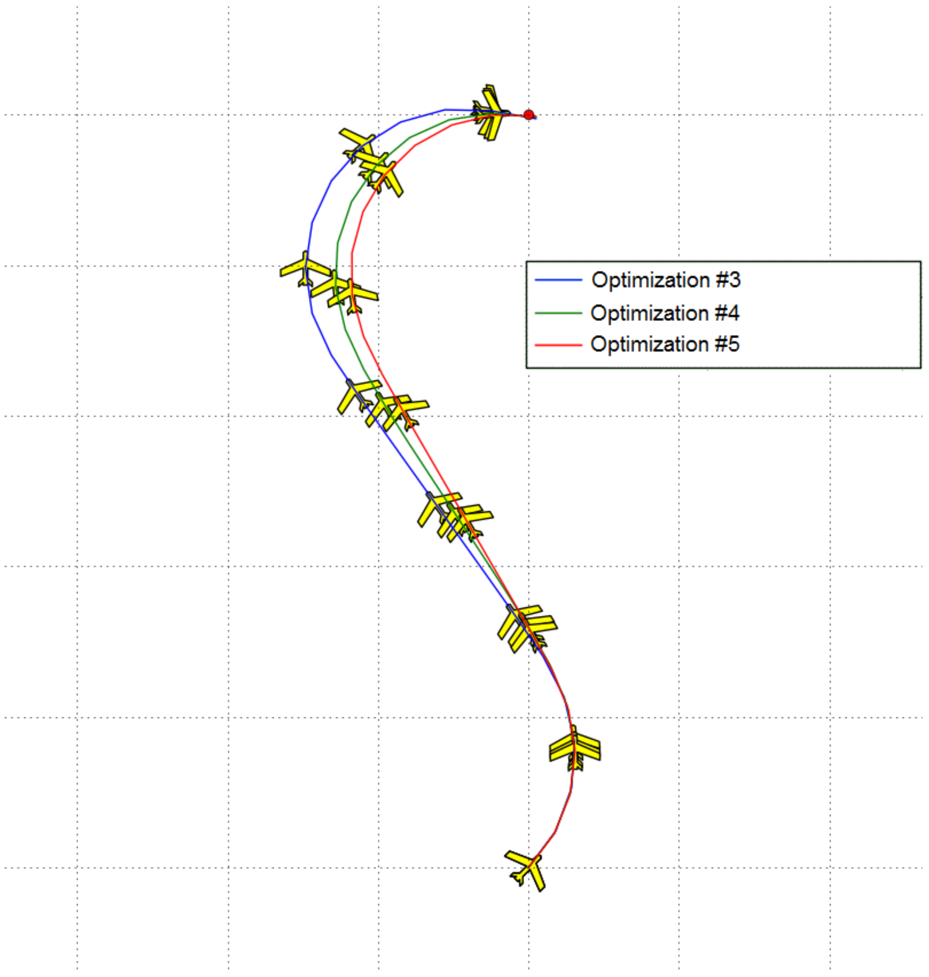

Figure 4-2: Bisection result on the path to the first waypoint. The optimization algorithm finds better solutions during each iteration. The two first optimizations were infeasible and are not shown.

4.1.3 **Wind compensation in the optimization algorithm**

Figure 4-3 shows a comparison of the optimized path for the UAV for a case with no wind, and a case with wind disturbance of $9m/s$, which is compensated for in the optimization.

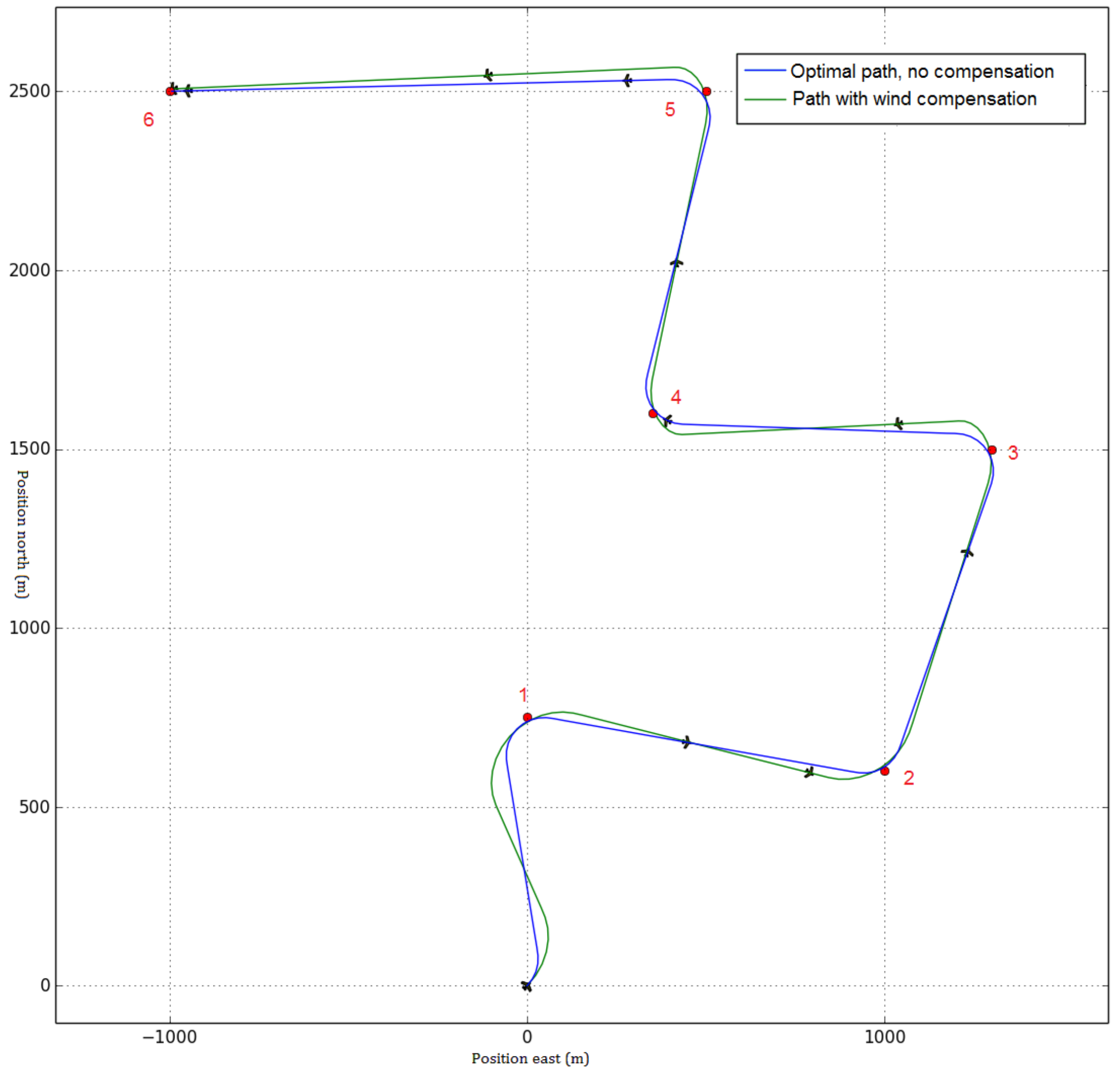

Figure 4-3: Optimized path with and without wind compensation (9m/s wind)

As shown in Figure 4-3, in the case with wind disturbance (green), the wind forces the UAV north-east. This makes the turns of the UAV appear sharper or wider compared to the undisturbed path, however, it reaches all the waypoints. Consequently, it appears that the controller is able to handle winds of this strength.

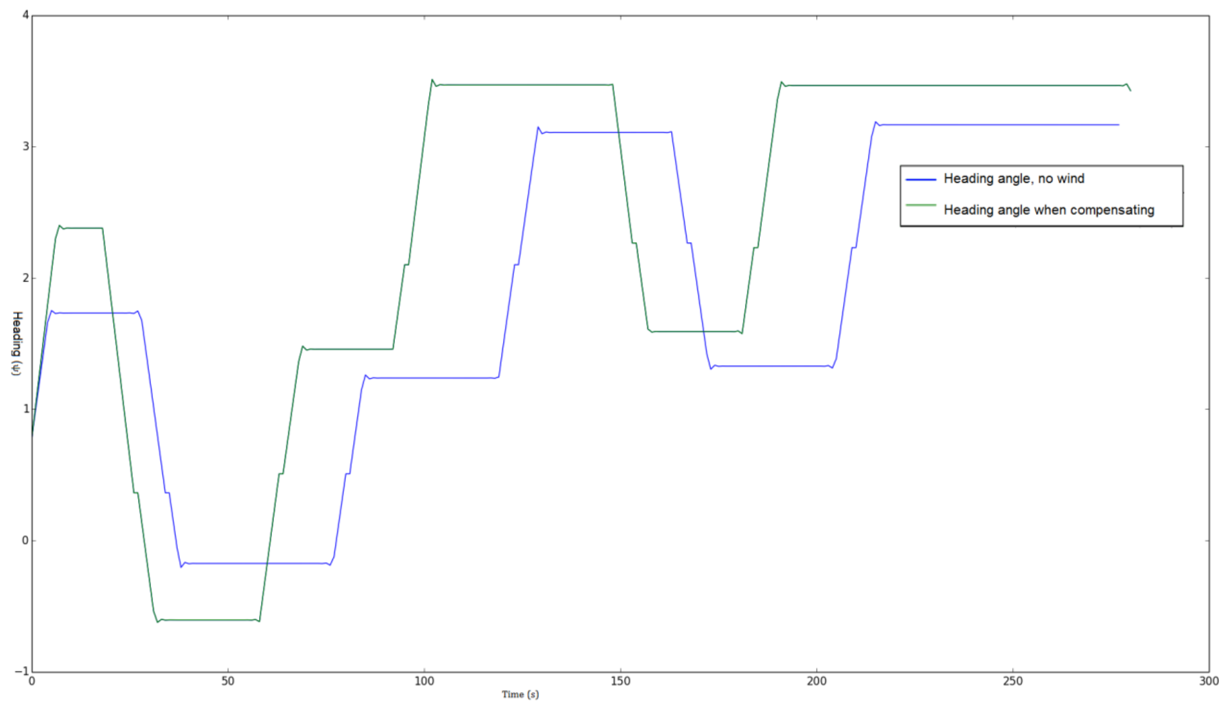

Figure 4-4: Optimized heading with and without wind compensation (9m/s).

In Figure 4-4, the heading angle for the optimized paths during the same scenario is shown. Here it is easy to see that the wind disturbance makes some of the turns faster than in the undisturbed scenario.

4.2 MPC RESULTS

For the MPC controller a series of simulations were run. The difficulty, i.e. the wind disturbance, was increased over the course of the tests. Starting off with no disturbances at all, to test the MPC under ideal condition, to scenarios with strong and changing winds. In the last scenario, the wind speed is more than half the speed of the UAV. An overview of the five main scenarios for the MPC is shown in Table 4-2.

Table 4-2: MPC test Scenarios

#Test \ Modes	Wind speed	Wind change	Wind compensation	Shown in:
Scenario 1	Off	Off	Off	Figure 4-5
Scenario 2	5 m/s	On	Off	Figure 4-6
Scenario 3	5 m/s	On	5 m/s	Figure 4-7 Figure 4-8
Scenario 4	9 m/s	On	9 m/s	Figure 4-9
Scenario 5	10+ m/s	Off	10+ m/s	Figure 4-10

4.2.1 Without wind disturbance

The first test scenario for the MPC controller was under ideal conditions. The main goal of this scenario is to test the MPC itself, and to investigate how the mismatch between the plant- and optimization model impacts the performance.

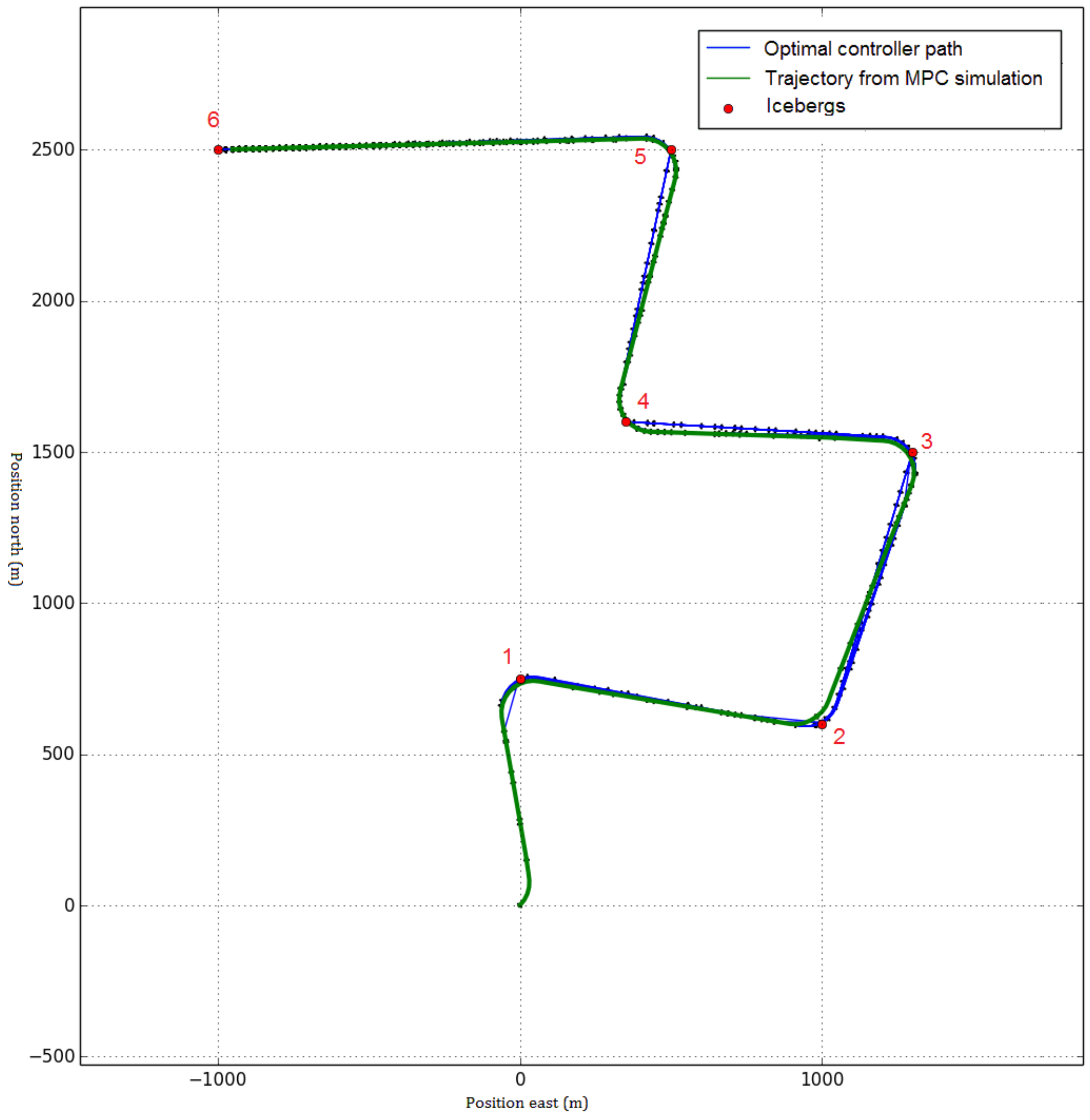

Figure 4-5: Scenario 1: MPC controller with no wind disturbance. Only the plant-model mismatch affects the solution.

As shown in Figure 4-5, the simulated UAV (green) follows the newest optimal path (blue) quite closely. At the first waypoint the UAV turns somewhat slower than the optimal path. This is mainly caused by the constraint placed on the bank-rate for the UAV in the simulation model. Close to waypoint two, the UAV finds a solution which takes it through the feasible region around the waypoint, but still close enough that it can go to the next one. At waypoint four, a non-optimal solution is found near the waypoint, most likely due to a poor time-approximation. This makes the UAV take an unnecessary large turn.

A major weakness of the optimal controller occurs when there is a short distance to the current waypoint. Then, the optimization does not find any feasible paths to the nearest waypoint. This makes the UAV use the inputs from the previous optimization until it has reached the waypoint and the next target is set. Nevertheless, in most cases this works out well, but in the scenarios with changing wind disturbance, the UAV might be blown off course while it uses the previous inputs and has no way of correcting its course. An example of this is shown later on.

4.2.2 With wind disturbance and no compensation

In the second scenario the wind disturbance is turned on. However, the optimization algorithm has no knowledge of this disturbance. The results are as expected, as shown in Figure 4-6.

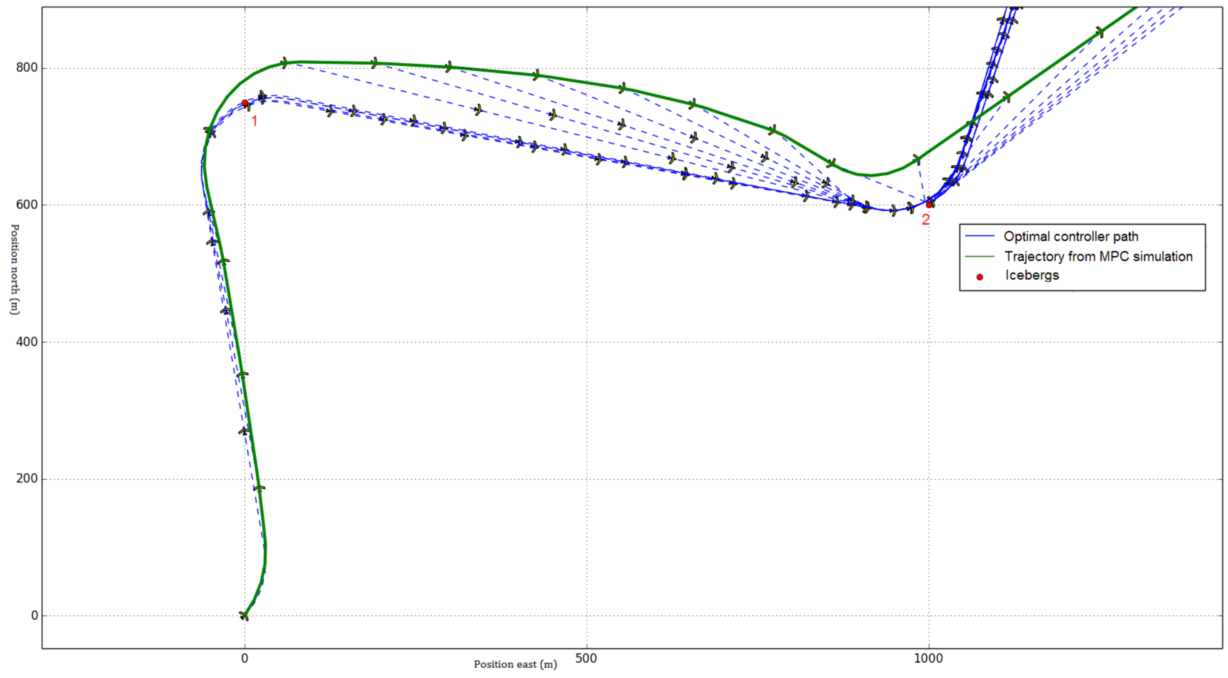

Figure 4-6: Scenario 2: Simulation with 5m/s wind disturbance and no compensation in the optimization model.

With no knowledge of the wind disturbance the UAV is blown out of course after reaching the first waypoint. The optimal paths (blue trajectories) found between the first and second waypoint try to steer the UAV back on course but since it cannot correct for the wind, it misses waypoint number two. Since no backup controller was implemented for the UAV in this project, the UAV continues to try to find a feasible path back to the waypoint, but finds only infeasible solutions.

4.2.3 Constant wind compensation, with changing disturbances

In this scenario, the wind disturbances on the simulation model and the wind used in the optimization algorithm starts out the same. However, the wind used on the simulation model is allowed to change from its initial value over the course of the simulation. This will give an error that is small in the beginning and grows over time. The net disturbance of the wind will be the difference of the compensated wind and the wind on the simulation model. The change in the wind was done by either increasing or decreasing the strength and direction randomly at each simulation step. Due to the random wind change, a number of different cases could be tested for by changing the randomization seed. How much the wind changed from its initial values, greatly impacted the solutions as seen in Figure 4-7 and Figure 4-8.

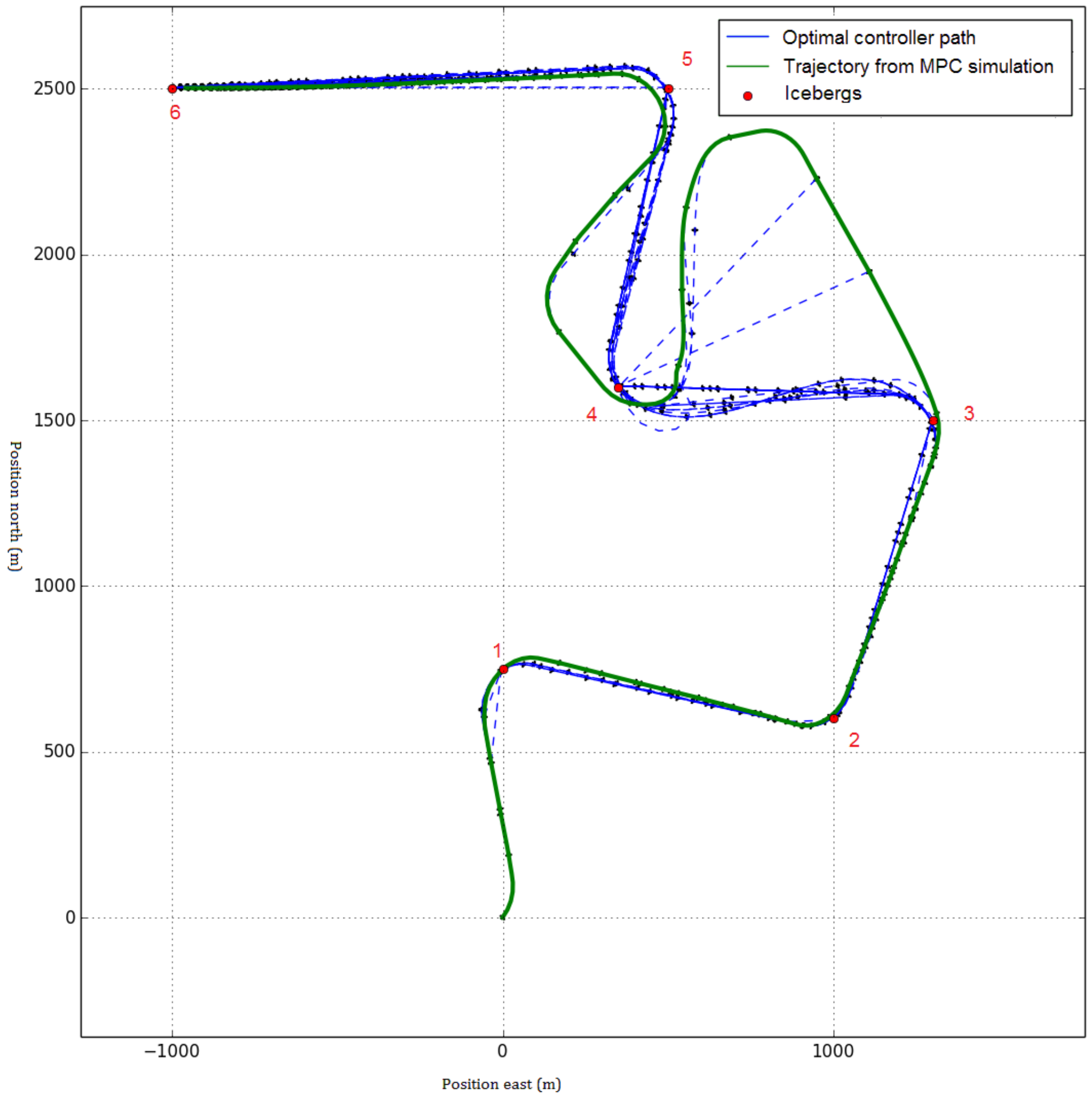

Figure 4-7: Scenario 3: Simulation with 5 m/s wind compensation, but the wind changes over time. Problems arise after third iceberg.

As seen in Figure 4-7, if the wind changes too much from the initial value, the UAV is unable to make the required turns, and therefore will drift off course. However, in this case it finds a new solution after going off course after iceberg number three, and manages to steer through the rest of the waypoints. Nevertheless, the overall solution is clearly suboptimal. The straight blue lines that go from the green path after iceberg three directly to waypoint four are infeasible solutions, and the optimization algorithm cannot find a solution with the time-horizon given by the bisection algorithm.

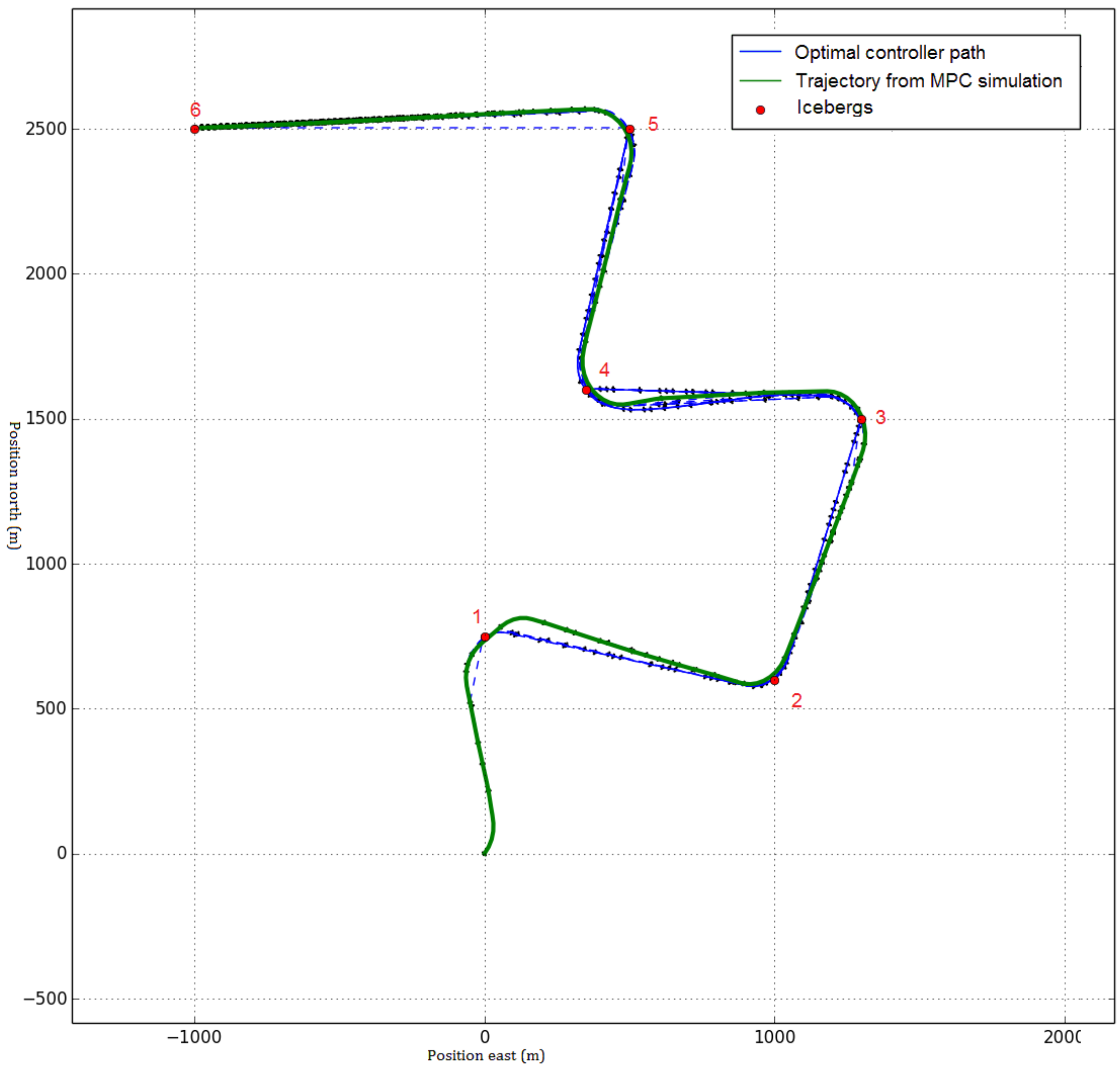

Figure 4-8: Scenario 3: 5m/s constant wind compensation. A solution where the wind changes are too small to impact the solution.

In Figure 4-8, a simulation is shown where the UAV reaches all the waypoints. Here the wind change was below $\pm 1\ m/s$ in magnitude and less than 20^o in direction from its original strength of 5 m/s. If the change in wind direction is small, the constant compensation in the optimization makes this scenario roughly equivalent to a scenario with just a wind disturbance of $1\ \mathrm{m/s}$.

However, if the change in wind direction is large, not only is the constant compensation incorrect. It may in the worst case induce an extra disturbance as strong as the wind itself, if the actual wind turns 180^o and comes from the opposite direction of the estimate.

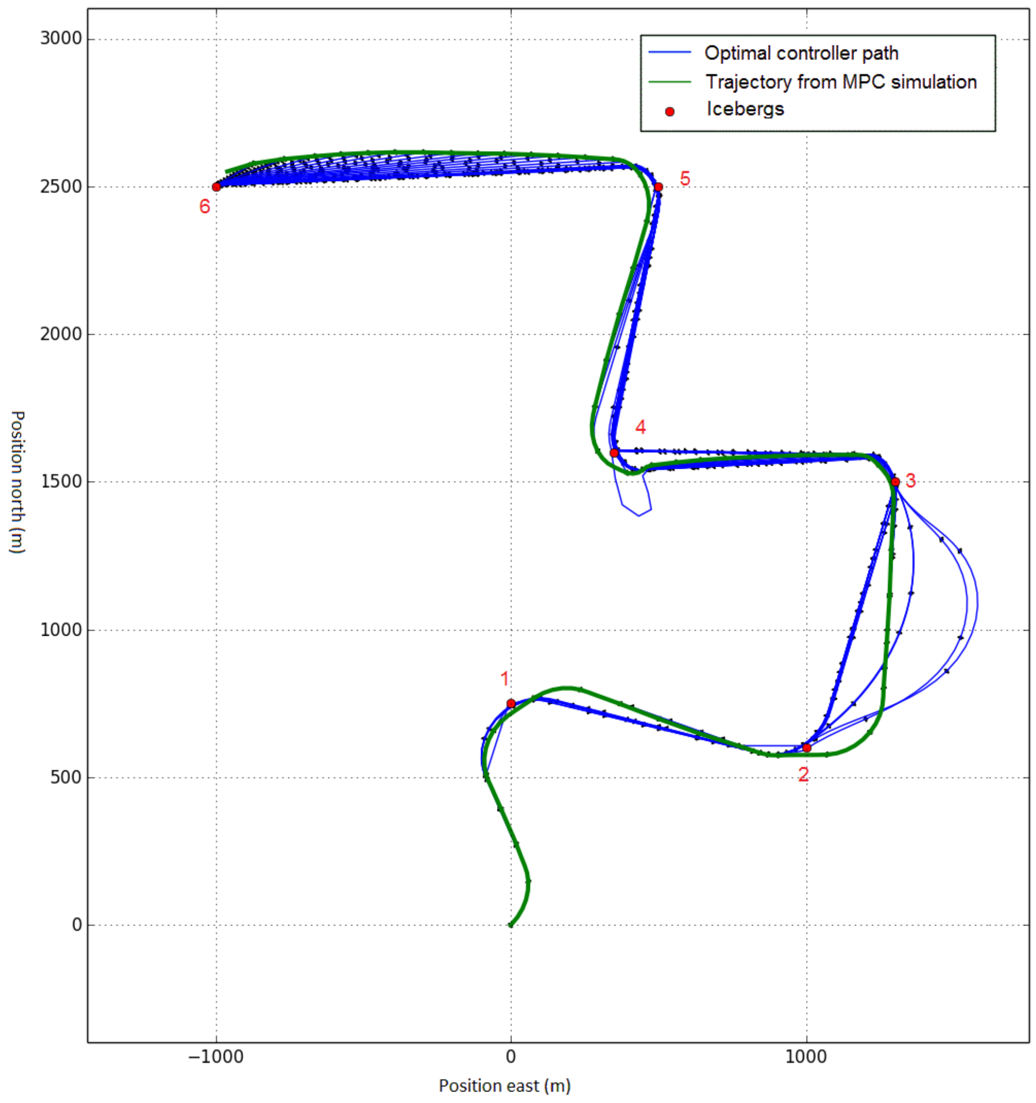

Figure 4-9: Scenario 4: 9 m/s with dynamic wind. The UAV is blown off course multiple times.

In Figure 4-9, one successful simulation with 9 m/s dynamic wind is shown. As in scenario 3, several simulations had to be completed to find a working solution. It is clear that the optimal controller does not work well if the wind information is outdated.

4.2.4 Performance with stronger winds

While the results may vary substantially when the wind is allowed to change, the MPC controller can handle quite strong winds when the actual wind is the same as the estimated wind. This is equivalent to a scenario where the actual wind is estimated perfectly during the flight, in which case the difference between the compensated wind and the actual wind is zero. With constant wind disturbance and the wind change turned off, the model predictive controller was found to handle wind strengths up to 12 m/s as shown in Figure 4-10.

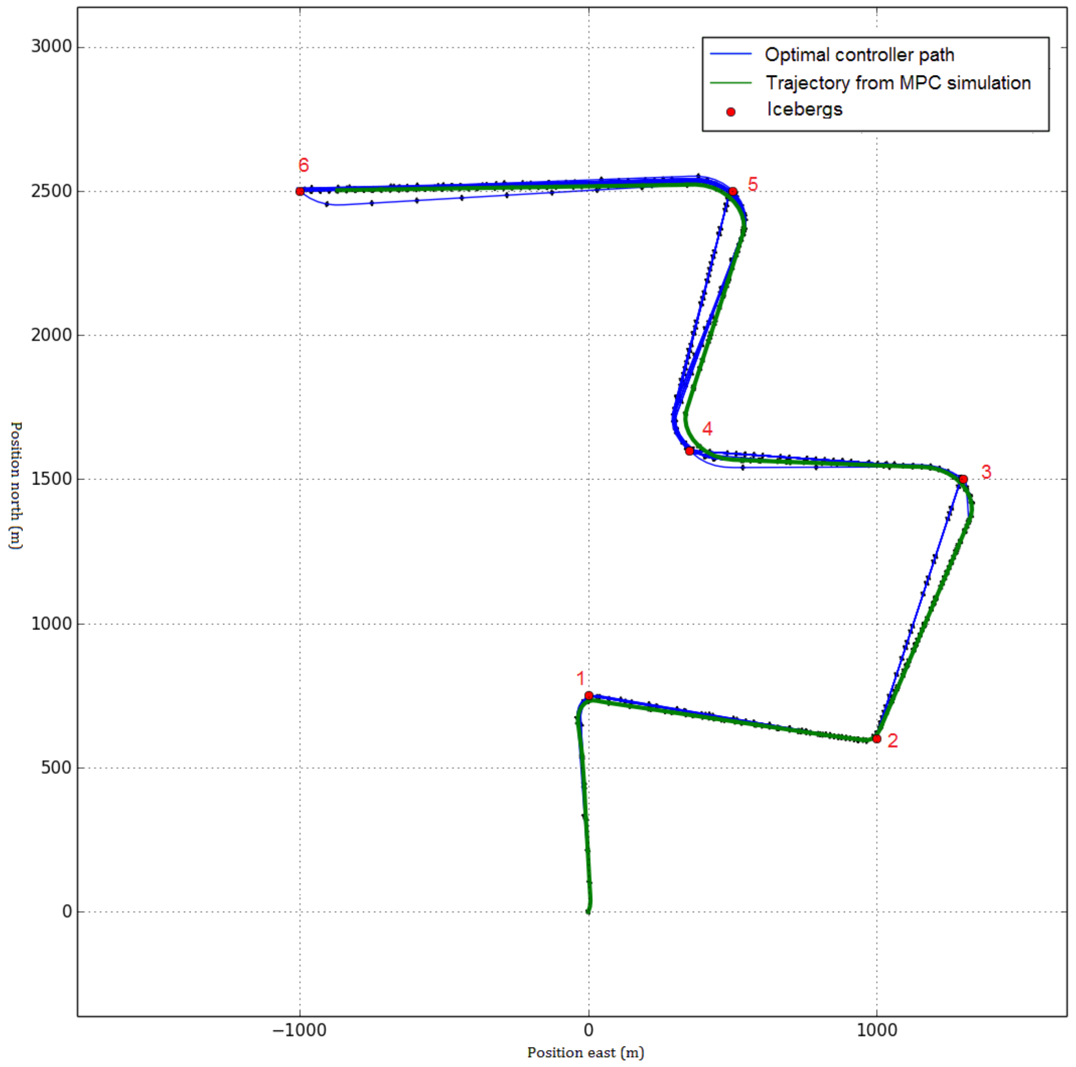

Figure 4-10: Scenario 5: Successful simulation with 12 m/s wind. To get this result the wind change had to be turned off.

5 DISCUSSION

First in this chapter, the optimal controller with the wind compensation and the model predictive controller will be discussed. Then some improvement for both the optimal controller and the MPC are suggested.

5.1 OPTIMAL CONTROLLER

5.1.1 The optimal controller

The optimization scheme used in this project succeeded in calculating a feasible path through all the waypoints. The solutions found were similar in both distance and form to the optimal Dubins paths.

The optimal controller had difficulties finding feasible solutions when close to the current waypoint. This is a result of how the starting time horizon for the bisection is chosen, and the fact that the limited turn-rate for the UAV makes the feasible region for the optimization algorithm smaller on shorter distances. A better approximation of the time-horizon and more dynamic time steps for the bisection might have resolved this. However, the problem will not disappear completely, due to the way the algorithm is built.

5.1.2 The bisection algorithm

The bisection algorithm worked well in finding an optimal time-horizon for the optimization. The method may not be the most efficient with respect to computational time, but it was suitable for finding an optimal time horizon, while keeping the optimization problem simple. One weakness of the bisection algorithm is that it is reliant on a good initial guess of the time horizon, which the bisection can operate around. This initial guess was based on an approximation of the time it should take to fly to the next waypoint, using geometry, UAV specifications, and weather information. Using this approximation method works in most cases, but may cause problems when the UAV is close to the target waypoint, or facing the wrong direction.

To achieve what is called a ‘warm start’ in optimization, a simpler and faster optimization problem could have been solved to find a better starting value for the time horizon, before initiating the main problem. However, the interior point methods are difficult to ‘warm start’. Furthermore, the performance of the bisection was deemed sufficient for the scope of the project.

5.1.3 Wind compensating

The addition of wind compensating to the optimization model worked well. It allowed the optimal control algorithm to take both wind direction and strength into account, and compensated for these when finding the control inputs for the system. However, since the wind compensating is constant for the duration of the optimization, any change in the wind after the start of the optimizations is not compensated. Including a wind estimator, such as a Kalman filter, that runs in the background, and updates the wind information, would be a good addition to the system. In this way, the model predictive controller would have updated wind information on each iteration.

The wind used in this project is modelled as strength and direction values that change over time. However, this wind model does not take gusts into account. Wind strength is usually modelled with two parts, one mean value part and one randomized part that will act as gusts and fluctuation of the disturbance. The simplification of the wind in this project to be just the mean value, is justified by the fact that a more complex wind model would not have influenced the optimization model, as the random disturbance from the gusts cannot be predicted. It could, however, been included in the plant model as additional disturbance, to make it more realistic.

5.2 MODEL PREDICTIVE CONTROLLER

Overall, the model predictive controller performed well. It was found that the UAV can handle wind with strength up to 12 m/s, which is more than half the speed of the UAV. This is when using the control inputs found by the optimal controller directly on the simulation model. The results show the strength of the MPC-scheme, where some of the disturbances can be coupled forward and compensated for in advance.

5.2.1 Wind change

The controller had, however, a few challenges. While the MPC can handle strong winds if the magnitude and direction is known, problems arise when the wind changes from its initially known values. The controller was found to cope better with uncertainties in wind strength, than uncertainties in direction. This was especially the case with strong winds, which is logical since an error in direction will induce a larger total error if the wind is stronger. In the simulations where the starting wind was 5 m/s, the wind direction could change up to 20^o from the initial direction, the UAV would still be able to correct the course after each optimization. In the simulations with stronger winds, the same change in wind direction would induce an error large enough to blow the UAV off course. A solution to the unknown wind components could be to include an estimator, such as a Kalman filter, to estimate the wind during the flight.

5.2.2 Infeasible solutions near waypoints

As mentioned in the section about the optimal controller, the optimization algorithm had trouble in finding feasible solutions when the UAV is close to the current waypoint. This problem occurred at almost all the waypoints, since the simulation time for each optimization was set to just 5 seconds.

During the time when no current feasible solution was found, the inputs from the previous successful solution was used until a new set of inputs was found. However, since the optimization model and the simulation model are different, the simulated UAV turns slower, and will need a course correction with the next optimization. If multiple infeasible solutions occur in sequence, the error would increase. This problem could be solves by implementing a backup path following controller, which could take control of the UAV when the optimization algorithm does not find a solution.

5.2.3 Missed waypoints

Yet another problem occurred if the UAV flied outside the accuracy margin around of one of the waypoints due to one of the two problems mentioned above. Since the speed of the UAV is constant and the turn-rate quite limited, the UAV will have to fly a quite large circle to visit the waypoints again. And the problem with inaccurate approximations of the time horizon, is even more clear when the UAV have to make a complete turn.

5.3 SUGGESTED IMPROVEMENTS

Changing winds was one of the major challenges for the MPC. Introducing a wind estimator such as a Kalman filter would most probably solve this problem. A Kalman filter estimates the change in the wind between the optimizations, so that each iteration of the MPC has a new estimate of the wind. As seen from Figure 4-10, the controller can handle quite strong winds as long as they are known, so this addition would probably contribute to a large improvement in performance.

Another possible improvement is to include a backup controller which takes control of the UAV in the cases where the optimization does not find a feasible solution. This controller could be a simple line of sight algorithm using the waypoints. However, a path following guidance system, using the last feasible solution, would probably be the best, since an optimal path is already generated. This backup controller might not be required if the wind estimator works sufficiently well.

6 CONCLUSION AND FURTHER WORK

6.1 CONCLUSION

The model predictive control scheme using end time bisection performs well for traversing a number of waypoint with an UAV. The MPC was able to compensate for relatively strong winds, as long as it had access to updated wind information. However, the bisection method had trouble finding suitable time horizons when the distance to the next waypoint was short.

6.2 FURTHER WORK

Further work to improve the controller may include:

- Improve the bisection method, to better find time horizons for the optimization.
- Expanding the controller with a Kalman filter to estimate the wind change over the course of the surveillance flight.
- Implement a backup controller that can take control over the UAV is the optimization algorithm does not find a feasible solution.

REFERENCES

1. *Scientific American.* March 1849
2. Cook, K.L.B. *The silent force multiplier: The history and role of UAVs in warfare*. in *IEEE Aerospace Conference Proceedings*. 2007.
3. Tucker, S. and P.M. Roberts, *The encyclopedia of the Arab-Israeli conflict : a political, social, and military history*. 2008, Santa Barbara, Calif.: ABC-CLIO.
4. Amazon.com. *Amazon Prime Air*. 2015 2015-11-30 [cited 2015 2015-12-7]; Amazon drone delivery system]. Available from: http://www.amazon.com/b?node=8037720011.
5. Berni, J.A.J., et al., *Thermal and narrowband multispectral remote sensing for vegetation monitoring from an unmanned aerial vehicle.* IEEE Transactions on Geoscience and Remote Sensing, 2009. **47**(3): p. 722-738.
6. Ryan, J.C., et al., *Repeat UAV photogrammetry to assess calving front dynamics at a large outlet glacier draining the Greenland Ice Sheet.* The Cryosphere Discuss., 2014. **8**(2): p. 2243-2275.
7. Eik, K.J., *Ice management in arctic offshore operations and field developments*. 2010, Trondheim: Norwegian University of Science and Technology, Faculty of Engineering Science and Technology, Department of Civil and Transport Engineering. V, 214 s. ill.
8. Rider, L., *Analytic Design of Satellite Constellations for Zonal Earth Coverage Using Iinclined Circular Orbits.* Journal of the Astronautical Sciences, 1986. **34**(1): p. 31-64.
9. Haugen, J. and L. Imsland, *Autonomous aerial ice observation for ice defense.* Modeling, Identification and Control, 2014. **35**(4): p. 279-291.
10. Skjetne, R., L. Imsland, and S. Løset, *The Arctic DP research project: Effective stationkeeping in ice.* Modeling, Identification and Control, 2014. **35**(4): p. 191-210.
11. Hektoen, N.M., *Optimal waypoint following for an UAV using end-time bisection.* 2015, NTNU.
12. Fossen, T.I., *Handbook of Marine Craft Hydrodynamics and Motion Control*. 2011, Chichester, West Sussex, U.K.: John Wiley & Sons Ltd.
13. Dubins, L.E., *On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents.* American Journal of Mathematics, 1957. **79**(3): p. 497-516.
14. Anderson, E.P., R.W. Beard, and T.W. McLain, *Real-time dynamic trajectory smoothing for unmanned air vehicles.* IEEE Transactions on Control Systems Technology, 2005. **13**(3): p. 471-477.
15. Nocedal, J. and S.J. Wright, *Numerical Optimization*. 2 ed. Springer Series in Operations Research and Financial Engineering. 2006: Springer-Verlag New York.
16. Diehl, M. *Numerical Optimal Control* 2011.

17. Andersson, J., *A General-Purpose Software Framework for Dynamic Optimization*. 2013, Arenberg Doctoral School, KU Leuven: Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium.

18. Griewank, A. and A. Walther, *Evaluating Derivatives*. Other Titles in Applied Mathematics. 2008: Society for Industrial and Applied Mathematics. 448.

19. Wächter, A. and L.T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*. Mathematical Programming, 2006. **106**(1): p. 25-57.

20. Przybylak, R., *The Climate of the Arctic*. 1 ed. Atmospheric and Oceanographic Sciences Library. 2003: Springer Netherlands.

21. *HSL. A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk/.*

22. Cormen, T.H., et al., *Introduction to Algorithms, Third Edition*. 2009: The MIT Press. 1312.

23. Albert., A. and L. Imsland. *Mobile Sensor Path Planning for Iceberg Monitoring Using a MILP Framework*. 2015.

APPENDIX

A. PYTHON CODE

Constraint generation:

```
#Collocated states and parametrized controls
for k in range(n):
     #collocated states
     for j in range(d+1):
          # Get the expression for the state vector
          X[k, j] = w[offset:offset+nx]
          vars_init[offset:offset+nx] = position
          if k ==0 and j ==0:
               vars_lb[offset:offset+nx] = xi_min
               vars_ub[offset:offset+nx] = xi_max
          else:
               vars_lb[offset:offset+nx] = x_min
               vars_ub[offset:offset+nx] = x_max
          offset += nx
     #Parametrized controls
     U[k] = w[offset:offset+nu]
     vars_lb[offset:offset+nu] = u_min
     vars_ub[offset:offset+nu] = u_max
     vars_init[offset:offset+nu] = u_init
     offset += nu
#State at end time
X[n, 0] = w[offset:offset+nx]
vars_lb[offset:offset+nx] = xf_min
vars_ub[offset:offset+nx] = xf_max
vars_init[offset:offset+nx] = position
offset += nx
```

Code A-1: Constraint generation

Bisection logic:

```
if exit_msg == "Solve_Succeeded":
      last_succ = t_const
      if last_fail != -inf:
            t_const = (last_fail + t_const) / 2
      else:
            t_const -= 3
elif exit_msg == "Infeasible_Problem_Detected":
      last_fail = t_const
      if last_succ != inf:
            t_const = (last_succ + t_const) / 2
      else:
            t_const += 3
elif exit_msg == "Invalid_Number_Detected":
      last_fail = t_const
      if last_succ != inf:
            t_const = (last_succ + t_const) / 2
      else:
      t_const += 3
elif exit_msg == "Maximum_Iterations_Exceeded":
      last_fail = t_const
      if last_succ != inf:
            t_const = (last_succ + t_const) / 2
elif last_succ == t_const:
      t_const += 2
      else:
            t_const += 2
elif exit_msg == "Restoration_Failed":
      t_const -= 1
      if last_msg == "Restoration Failed":
            t_const += 2
```

Code A-2: Bisection logic.

Runge-kutta integrator:

```
def rk4(x0, u, w, T, N, f):
     x = x0
     DT = T/N
     for k in range(N):
          k1 = DT*f(x, u, w)
          k2 = DT*f(x + 0.5*k1, u, w )
          k3 = DT*f(x + 0.5*k2, u, w )
          k4 = DT*f(x + k3,u, w)
          x = x + (k1 + 2*k2 + 2*k3 +k4)/6
     return [round(x[0], 3), round(x[1], 3), round(x[2], 5) ]
```

Code A-3: Runge kutta integrator: