NTNU – Trondheim
Norwegian University of
Science and Technology

UAV Path Planning for Ice Intelligence Purposes using NLP

Lars Arne Grimsland

Master of Science in Engineering Cybernetics
Submission date: June 2012
Supervisor: Lars Imsland, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Engineering Cybernetics

PROJECT DESCRIPTION SHEET

Name of the candidate: Lars Arne Grimsland

Thesis title (Norwegian):

Thesis title (English): UAV path planning for ice intelligence purposes using NLP

Background

As offshore oil- and gas production enters arctic seas, the presence of ice becomes a substantial challenge in dynamic positioning of vessels. An important part of a DP system in the arctic, is a system for doing "ice management", which likely will include the use of image/other measurements from UAVs and sensors placed on the ice by e.g. UAVs to gather ice measurements. This master project will look at how to do path planning for UAVs using nonlinear programming (NLP). The master project builds on project work fall 2011.

Work description

1. Give a brief overview over the challenges with DP in ice, and how UAVs can be of help in this regard.
2. Describe how NLP and collocation can be used for (online/offline) path planning.
3. Implement a framework for UAV path-planning based on NLPs and collocation, using a NLP solver of choice (for example IPOPT). Discuss numerical efficiency, especially exploitation of problem structure, and make implementation. Illustrate and discuss optimizations/solutions.
4. Use the framework to suggest and make test implementations of solutions (including objective functions) for ice intelligence purposes. Discuss different objectives, like ice berg tracking (one or more ice bergs), and “zone surveillance”.
5. Suggest (and test) online/recursive path planning based on the implemented framework.

Start date: 9 January, 2012 **Due date:** 4 June, 2012

Supervisor: Lars Imsland
Co-advisor(s):

Trondheim, __09.01.2012_____

Lars Imsland
Supervisor

Address Sem Sælandsvei 5 NO-7491 Trondheim
Org.no. 974 767 880 E-mail: postmottak@itk.ntnu.no http://www.itk.ntnu.no
Location O.S. Bragstads plass 2D NO-7034 Trondheim
Phone + 47 73 59 43 76
Fax + 47 73 59 45 99
Phone: + 47 47 23 19 49

Abstract

As the oil and shipping industry are interested in operating in arctic waters, the need for ice intelligence gathering is rising. The thesis describes the implementation and results of a path planning framework for an UAV used for ice intelligence purposes. The framework produces paths based on optimization of a non-linear problem, using the IPOPT library in C++. A model for information uncertainty is implemented, and optimal paths based on minimizing the total information uncertainty are compared to optimal paths based on minimizing distance between UAV and target. Both off line and off line path planning is tested with single and multiple targets.

It was found that minimizing information uncertainty can work very well for path planning for ice berg surveillance, or for surveillance of a small search grid. Minimizing information uncertainty generally gave better results than minimizing distance between the UAV and given targets.

The implementation should be made more robust, and interfaces towards other UAV systems has to be made before the path planning platform has any practical use.

Sammendrag

Ettersom både olje- og shippingindustrien øker aktiviteten i farvann med mye is, har bedre metoder for overvåkning av is blitt etterspurt. Denne masteroppgaven presenterer implementasjon og resultater med et rammeverk for optimal ruteplanlegging for en UAV brukt til overvåkning av isfjell. Rammeverket baserer seg på optimalisering og bruker IPOPT, et bibliotek til C++, for å løse et ulineært optimaliseringsproblem. En modell for informasjonsusikkerhet i et gitt punkt er implementert, og optimale ruter basert på minimering av totalt usikkerhet blir sammenliknet med ruter basert på minimering av total avstand mellom UAV og punktet. Tester er utført med et og flere punkter som skal overvåkes, og rammeverket er utvidet til også å oppdatere rutene, som en modellprediktiv regulator.

Det ble funnet at minimering av usikkerhet kan fungere meget godt til ruteplanlegging for isfjellovervåkning, eller for overvåkning av et mindre søkeområde definert av et sett punkter. Minimering av informasjonsusikkerheten i gitte punkter fungerte generellt bedre enn minimering av avstand til gitte mål. Denne implementasjonen trenger tiltak for å bli mer robust, og også tilknytningsmuligheter til andre UAV-sytemer før det evt kan brukes i praksis.

Preface

You are now holding my Master's Thesis, which marks the end of my studies at the Master Program in Engineering Cybernetics at NTNU. The thesis is worth 30 credits, and it has been a full time occupation this last semester.

While writing the thesis, I have assumed that the reader have some knowledge of optimization theory, MATLAB and C++ programming. I've tried to refer to good textbooks on the subjects discussed as well, so the interested reader can find more background information.

As this master thesis is a continuation of a fall project, some parts in the background and theory chapter, will bear resemblance to the fall project report. I choose to include edited versions of these parts instead of having longer citations or references to increase the readability of the report.

Thanks to Professor Lars Imsland for giving me an interesting thesis, and keeping the office door open. That you have been interested in my work, taken time to listen to the challenges met and given me pointers on how to get passed them have been much appreciated.

I'd also like to thank Post Doc. Esten Grøtli and Ph.D. student Joakim Haugen for giving me useful pointers on the uncertainty model together with Prof. Imsland.

I must also thank all my friends at the university for all the coffee brakes, bad jokes and good discussion. All of them were needed, and it's been a blast!

Last, but not least, a big thank you to my fiancé Live for your patience, support and encouragement. Now I'm coming home to marry you!

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations are listed in table 1. Abbrevatios will also be explained in the text where first used.

Abbrevations	Full expression w/explantion
GCS	Ground Control Station
IPOPT	Interior point optimiser. A algorithm/software library for solving NLPs.
MUMPS	Multifrontal massively parallel sparse direct solver. Software package for solving linear systems.
LP	Linear Program. Defines a liner optimization problem.
NLP	Nonlinear Program. Defines a non linear optimization problem.
ODE	Ordinary Differential Equation.
SQP	Sequential Quadratic Program.
UAS	Unmaned Aircraft System.
UAV	Unmaned Aereal Vehicle.
QP	Quadratic Program. Defines a quadratic optimization problem.

Table 1: Abbreviations

Symbols and notations are explained in table 2. Symbols and naming is also explained where first used. 0-indexing of all variables is chosen, due to the equations and variables are to be implemented in c++. C++ uses 0-indexing of all arrays and matrices.

Symbols	Explanation
d^s	defect vector of segment s. Dimension is $[n, 1]$
d_i^s	i-th element of vector d^s
g	vector of constraints. Dimentions is $[n \times S, 1]$
g_i	i-th element of vector g
m	number of inputs in a state space model
n	number of states in a state-space model. Also a given node in collocation scheme
n_i	a given node, the i-th. Nodes are zero-indexed; $i \in [0, S]$.
N	Total number of nodes. $N = S - 1$.
S	Total number of segments
s	a given segment. Segments are zero-indexed; $s \in [0, S - 1]$.
u	input vector of a state space model. Dimention is $[m, 1]$
u^j	input vector of the j-th node.
u_i	i-th element of vector u. Inputs are 0-indexed for consistancy with a c++ implementation; $i \in [0, m - 1]$
x	state vector of a state-space model. Also optimization variable vector in IPOPT library.
x^j	state vector at the j-th node
x_i	i-th state in a state space model / element of x-vector. States are 0-indexed; $i \in [0, n - 1]$. (Note that state variables are normally 1-indexed in the litterature, but 0 indexing is chosen, such that the naming will be consistent with a c++ implementation.)
y	dual optimization variable vector
y_i	i-th element of vector y
z	(primal) Optimimization variable vector.
z_i	i-th element of z vector. z is zero-indexed: $i \in [0, (n + m) * S - 1]$

Table 2: Nomenclature: symbols, naming and syntax.

Chapter 1

Background

In 2008, a study performed by the United States Geological Survey (USGS), estimated that there could be 90 billion barrels of oil, $4.7x10^{13}$ cubic meters of natural gas, and yet another 44 billion barrels of natural gas liquids in the areas north of the arctic circle [27]. At the time, these resources counted for about 22 % of the undiscovered, technically recoverable oil and gas resources in the world, with most of these located offshore.

With continuously high oil prices, and the world still being dependent on fossil fuels for years to come, it is no surprise that many of the worlds leading oil companies are turning their attention to the north. However, in these areas they are met with new challenges such as very low temperatures and extreme weather conditions, remoteness from any existing infrastructure and aids available for clean-up after spills, and last but not least; drift ice[20].

At the Marine Technology Society's Dynamic Positioning Committee's DP conference in Huston, Texas in October 2009, several papers regarding DP in the icy, arctic waters where published [2]. This reflects the interest for development and ongoing research in this field.

The Norwegian University of Science and Technology, NTNU, also participates in a research project related to Dynamic Position in the arctics, with partners such as Kongsberg Maritime, Statoil and DNV[4].

1.1 Dynamic positioning in sea ice

Station keeping is essential for any vessel performing a drilling operation. Using dynamic positioning is a popular method to obtain this. The Norwegian classification society, DNV defines the a dynamically positioned vessel as[14]:

> *...a free-floating vessel which maintains it position (fixed location or predetermined track) exclusively by means of thrusters.*

Traditionally, dynamic positioning (DP) is understood as using thrusters, and sometimes also the vessels rudders and propellers (if present) to control the horizontal movements; surge, sway and yaw, of a vessel. More recently there has also been suggested to

extend the DP system to give additional damping in the vessels roll and pitch movements [14]. This is favorable, as it not only is important for a drilling vessel to hold it's position above a well on the ocean floor, but also to have only small movements in roll and pitch to avoid too much stress on the drilling string. See also Figure 1.1 for the definition of a ships axis.

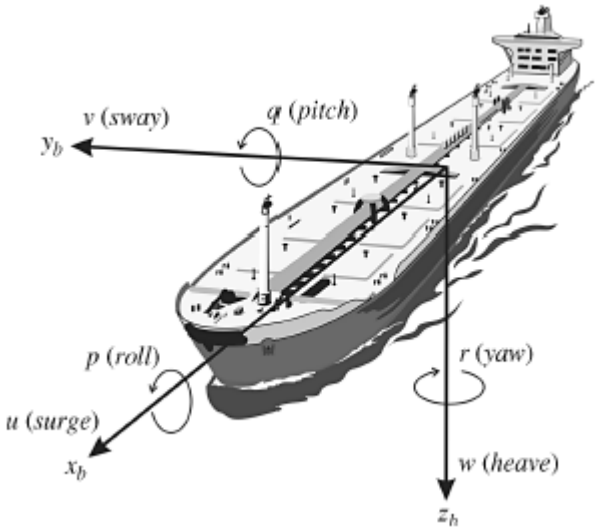


Figure 1.1: Definition of ship axis; surge, sway, heave, and rotation about these axis; roll, pitch, yaw. Illustration from Fossen[14].

Using a DP system is usually more convenient than anchoring, which demands use of specialized vessels for handling the anchors. But as position holding in ice only recently has become an issue, there is no commercial DP system available which is able to cope with the disturbances from ice[26].

An impact with an ice berg, or huge thick flakes of ice will produce a huge peak in the forces acting on a hull. Todays vessels does not have enough power available from their thrusters to counter the forces in such a collision.

A steady stream of smaller pieces of ice will act more like a disturbance caused by heavy currents, such that huge peaks are avoided. But this is also a challenge as the forces will still be significant. Still, small chunks are considered easier to handle than large flakes.

Adding to the complexity of the problem with ice is that ice drift is difficult to predict accurately. As of this, the moment of impact and the direction of forces is hard to predict. It is also important to be aware that collisions with, and pressure from ice could be strong enough to make dangerous structural damages to hulls, and even sink floating structures. Pressure from surrounding ice could also push a hull upwards, possibly causing instability, or damage to rudders, propellers and truster if they are hit by ice.

The mentioned observations draws out two possible solutions to cope with the sea ice, one not excluding the other. This is making DP systems more responsive, and keeping

dangerous ice away from the vessels. For instance: In a paper, Jenssen et.al [21] points at the need to make current DP systems more reactive, as ice loads to a vessels hull will be highly varying, with sudden high peaks. That current commercial DP systems are by now not responsive enough, is illustrated by the need to use manual joystick control of the DP system in the IODP 3021 core drilling expedition in 2004. The paper also concludes that more responsive DP systems will have to be combined with additional ice management.

To help the DP system, ice management involving ice breakers to break the ice into smaller pieces before it reaches the vessel is a feasible solution[21][26][24][20].

During a drilling maneuver, the tolerance of drift is often as little as a radius of 5 meter. Stability and station keeping is very important.

1.2 Ice management

Ice management can be defined as [17]:

> *Ice management is the sum of all activities where the objective is to reduce or avoid actions from any kind of ice features. This will include, but is not limited to:*
>
> - *Detection, tracking and forecasting of sea ice, ice ridges and ice bergs*
> - *Threat evaluation*
> - *Physical ice management such as ice breaking and ice berg towing*
> - *Procedures for disconnection of offshore structures applied in search for or production of hydrocarbons*

As mentioned, breaking up ice is only a part of ice management. Ice management is a complete system to deal with the ice; from planning and safety/operation manuals, to monitoring ice and predicting the ice drift through observations, measurements and weather data, to directing the ice breakers[24]. Figure 1.2 illustrates a possible realization of a sector based ice management system. Three ice breakers assists a drilling vessel using DP for station keeping.

Through monitoring and forecasting ice movements, the ice breakers are working in their individual sectors of a funnel shaped area, reaching out from the drilling vessel heading for the opposite direction direction of the ice drift. The biggest ice breaker works furthest out, to break up the biggest flakes of ice, which can be as big as a county. One by one, the ice breakers will break flakes of ice into smaller chunks, until the chunks are small enough for the drilling vessel to handle. As mentioned previously, it is essential that the ice chunks are as small as possible when reaching the position holding vessel for todays DP system to have a chance at working.

Rohlen describes briefly how this kind of ice management has been applied during two drilling expeditions; The Arctic Coring Expedition, drilling at 87.54 degrees north in August 2004, and the Kanumas Expedition drilling at 77 degrees north in July 2008[24].

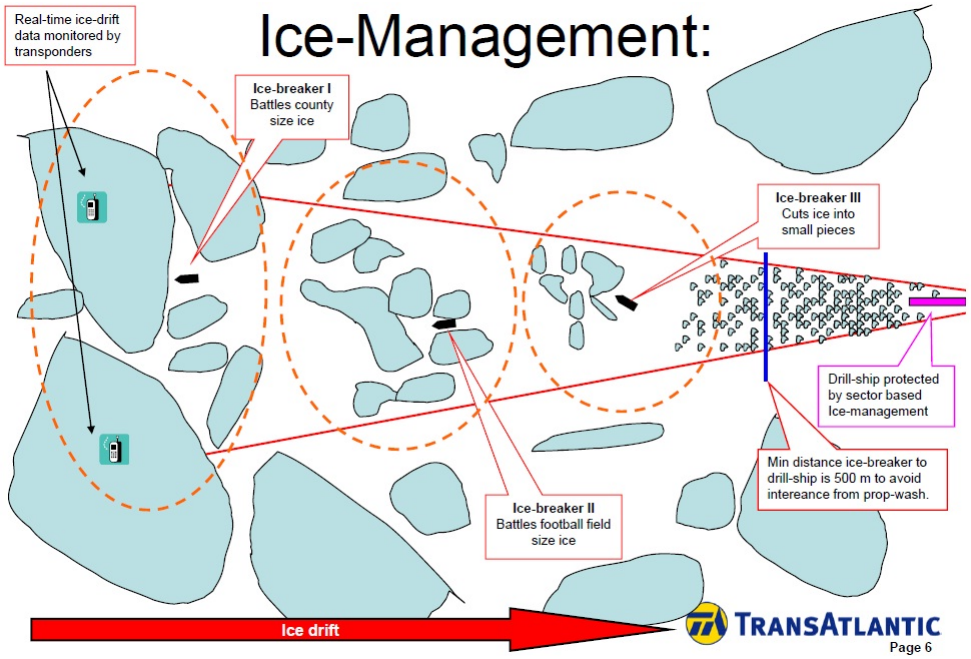


Figure 1.2: Illustration of a possible ice management system with three ice breakers assisting a DP vessel, as proposed by Rohlen[24]. Image is copied from Rohlen's presentation.

In the latter expedition, they were actually able to use the drilling vessels DP 80 percent of the time, in what was considered to be light ice conditions. But the conclusion is still that to enable position holding using dynamic positioning in arctic waters, DP systems must be more robust for ice pressure. Ice management must be implemented as discussed[21][24]. And for ice management to be effective, good measurements of the current ice conditions are a must, along with the need to predict changes in it. Knowledge of the ice flow is essential to direct ice breakers in the most efficient way.

It should also be noted that ice management not only applies to assisting DP vessels, but it can be used to assist any vessel or installation dealing with ice. Another upcoming area for ice management could for instance be assisting freight vessels and tankers wanting to travel through the North-West Passage.

1.3 Sensor platforms for ice monitoring purposes

So how do one keep track of ice? As already discussed, detection, tracking and forecasting of ice is an important part of ice management in order to direct ice breakers or to know when it is time to shut down drilling operations. Because of the highly complex behavior of ice drift, one has not been able to create highly accurate models or estimators for ice, neither for movement or thickness. This means that without good real time data to correct the estimates, the estimators will drift away from the correct states. There are

several ways to obtain these real time data.

In Figure 1.2, it is proposed to use transponders placed on ice flakes to monitor ice drift. These could be placed using helicopters. Measurement using radars mounted on vessels is another option, as well as using observations from vessels, helicopters or scout planes, or even using images and radar data from surveillance satellites. Haugen et.al. [17] gives a short evaluation of the available sensor platforms for ice monitoring. It is shown that unmanned aerial vehicles, UAVs, can be particularly useful as sensor platforms in an ice observation system in connection with ice management.

1.3.1 Unmanned Aerial Vehicle, UAV

An UAV is defined as a

> *powered, aerial vehicle that does not carry a human operator, uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload [3].*

There are many different UAVs, ranging from cheap and simple radio controlled helicopters, to extremely sophisticated autonomous planes used as sensor and weapon platforms. With the resent years technological advances, UAVs can replace human operated helicopters and surveillance planes, performing task that are "to dull, dirty or dangerous" for humans.

Along with the UAV, especially the autonomous ones, the term Unmanned Aircraft System is often used. It can be defined as:

> *...UAS is a system, which includes the unmanned aircraft / UAV, the launch and retrieval system, the ground control station (GCS) and the communication channel between the unmanned aircraft and GCS [25]*

UAVs are flexible on areas such as geographical coverage and resolution. For Ice monitoring purposes, an UAV could be equipped with photo/video cameras, Thermal/Infrared sensors,laser scanners, laser altimeters, radar altimeters etc. A flexible platform such as an UAV can provide data from areas both in the immediate proximity of the ice management vessels, or from far away [17].

Using an autonomous UAV in an ice observation system could increase the systems efficiency, as it could minimize the need for human participation in both planning and execution of a flight. A fully autonomous UAV should be able to take off, land and follow a flight path automaticly.

To operate an UAV, one will typically have the following systems

- Guidance system
- Control system (Autopilot)
- Navigation/Observer system

How these are interconnected is shown in Figure 1.3. The guidance system will typically produce the flight trajectory, and send set points to the control system. The trajectory should reflect the purpose of the flight, UAV capabilities and so forth. The control system (autopilot) will compare the given set points to the measurements of its position, velocities, etc. given from the Navigation/Observer system, and calculate outputs given to the UAV's actuators (motors, rudders, etc). The navigation system is made up of the devices performing measurements of the UAV states, and any filters or observers used to provide estimates of the states.

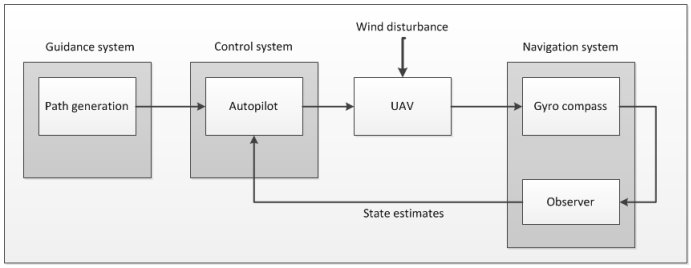


Figure 1.3: Interconnection between guidance system, autopilot and navigation system for an UAV.

1.4 UAV/UAS path planning

Whether or not the AUV is autonomous, it is obvious that it should follow some flight trajectory that gives good sensor coverage of a specific target or an area of interest. This path planning could be posted as a a mathematical problem, and a solution that is optimal and "best possible" in some sense could be computed.

Aspects to consider in this computation are what maneuvers the AUV is physically capable of and how it affects its performance and flight economy. One must also consider sensor coverage; namely what one want to gather information of, and what UAV behavior will give the best information gathering.

1.4.1 Off line path planning

In off line path planning, the guidance system produces a flight trajectory once, prior to the flight. An off line trajectory generation will typically be done by a GCS, and the path information is transfered to the autopilot. In later sections, a theoretical background for path planning using optimization theory will be given, and an implementation will be described and discussed.

1.4.2 On line path planning

In an on line path planning scheme, the trajectory is recalculated or updated during the flight, according to UAV's behavior, or as new information that might change the

flight objective is gathered. On line path planning using optimization falls under the domain of Model Predictive Control, which will be presented in later chapters. A simple implementation will also be described. An off line trajectory generation will typically be done by a GCS, while an on line could be done either by an onboard computer or a GCS which then transfers the path to the UAV.

1.5 Previous work

This thesis is a continuation of a 7.5 credits fall project. In the fall project a simple framework for optimal path planning was implemented in MATLAB, using the *fmincon* algorithm for sequential quadratic programming. The fall project was limited to off line path planning, minimizing the distance between an UAV and a single target. A collocation method was used to discretize the system.

1.6 Scope of thesis

The formal Project Descripiton Sheet given by the supervisor for this thesis is included on the first page of this document. The scope of the thesis is to:

- Give a brief overview over the challenges with DP in ice, and how UAVs can be of help in this regard.
- Describe how NLP and collocation can be used for off line and on line path planning.
- Implement a framework for UAV path-planning based on NLPs and collocation, using a NLP solver of choice.Discuss numerical efficiency, especially exploitation of problem structure, and make implementation. Illustrate and discuss optimization solutions.
- Use the framework to suggest and make test implementation of solutions (including objective functions) for ice intelligence purposes. Discuss different objectives, like ice berg tracing (one or more ice bergs), and "zone surveillance."
- Suggest and test on line/recursive path planning based on the implemented framework.

1.7 Contribution

There has been developed many path planning schemes, and some of these are also based on optimization. As optimization using nonlinear programming with collocation methods has seen limited use in UAV application [10], it is of interest to see how such an approach performs when computing an optimal path.

The contribution of this report is to describe how nonlinear programming and collocation can be used to calculate optimal paths for ice intelligence purposes. Two different

principles for designing objective functions for surveillance, namely distance minimizing and information uncertainty minimizing will be tested and discussed.

Using a model for information uncertainty is a new approach, and it's performance in an path planning for surveillance is of interest. Some similar models for probability and uncertainty has been used in other work, but not in connection with path planning as in this implementation. Therefore, giving examples of using this information uncertainty model for both off and on line path planning with single and multiple targets is the main contribution of this thesis.

MPC controllers are increasingly popular in the process industry, but is not so common for recursive path planning. Two concepts for on line path planning and MPC control is tested and discussed. The thesis contributes with demonstrations and discussions of possible uses.

Chapter 2

Theory

2.1 Optimization

In a mathematical context, optimization is to maximize or minimize a function, where this function has a set of variables which in turn is subject to a set of constraints. More practically put, is to find a best possible solution to a some function describing a systems behavior, within the systems constraints. This will require[9]:

- An objective function, which gives a scalar quantification of the systems performance.
- A predictive model, to describe the behavior of the system, and to express the limitations or constraints of the system.
- A set of variables that appear in the predictive model.

The concept of optimization is to find the combination of values for these variables that satisfies the constraints given by the predictive model and at the same time maximizes or minimizes the objective function. The general optimization problem with constraints can be expressed as follows[22]:

$$\min_{z} f(z) \tag{2.1}$$

$$s.t. \begin{cases} c_i(z) = 0 & i \epsilon \mathcal{E} \\ c_i(z) \leq 0 & i \epsilon \mathcal{I} \end{cases}$$

$f(z)$ is the objective function, which is also called the cost function when it is minimized, as in the general statement. *s.t.* stands for "subject to". $c_i(z)$ are the constraints. The constraints can be either equality or inequality constraints. The indexes i are either in set $\mathcal{E}$, which are the equality constraints, or in $\mathcal{I}$, which are the inequality constraints. z is the vector of the optimization variables, such as all system states, any control variables, etc.

2.1.1 Optimization Algorithms

There is no single, general algorithm which can solve the general optimization problem proposed in equation (2.1). Over the years, there has been developed a variety of algorithms, each suitable to solve a special version of this problem. Which algorithm to use, depends on the stated problem. Some algorithms works only for unconstrained problems. When both objective function and constraints are linear in z, an algorithm based on Linear Programming will typically be used. If the object function is quadratic in z and the constraint are linear, one will typically use a Quadratic Programming algorithm[22]. When the optimization problem is stated with nonlinear objective function and/or nonlinear constraints, Nonlinear Programming (NLP) is used. Good textbooks on optimization describing the mentioned algorithms and necessary theory are Nocedal and Wright's Numerical Optimization[22] and Biegler's Nonlinear Programming[9].

There are several NLP methods available. Sequential Quadratic Programming (SQP) and Interior Point methods are the most popular. This is because they are the most powerful algorithms for large scale nonlinear programming, and can handle significant non-linearities in the constraints and objective functions[22].

2.2 Interior Point methods

Interior point methods (sometimes called barrier methods) is an alternative to the SQP methods. A general "rule-of-thumb" based on numerical experience is that Interior Point methods tend to be faster than SQP methods when the non-linear problems are large and particularly when the problem has a large number of free variables [22]. The IPOPT library for c++ implements a Dual-Primal Interior Point method, for which the basis is outlined in the following. The interior point method solves the NLP by computing approximate solutions for a sequence of barrier problems [7]. The key advantage of the algorithm is that it is not needed to make decisions about active sets, which the SQP (Active Set) algorithms does [9]. On the other hand, Interior Point methods might be less robust[22].

2.2.1 The primal-dual barrier approach

Any general, nonlinear problem can be rewritten to the following, introducing slack variables if necessary:

$$\min_z f(z) \tag{2.2}$$

$$s.t. \begin{cases} c(z) = 0 \\ z \geq 0 \end{cases}$$

The solution to this general problem is found by solving a sequence of barrier problems

$$\min_z f(z) - \mu \sum_{i=1}^{n} ln(z) \tag{2.3}$$

$$s.t. \begin{cases} c(z) = 0 \end{cases} \tag{2.4}$$

The barrier function

$$\varphi_\mu := f(z) - \mu \sum_{i=1}^{n} ln(z) \tag{2.5}$$

is introduced because it quickly increases to infinity as z approaches the boundary of the feasible region. The algorithm is thereby given a stronger incentive to stay within the feasible region, as the search directions will be forced inwards[22]. The logarithmic term is preferred, because it can be differentiated twice quite easily. On the other hand, it might cause difficulties if the optimal point is on the edge of the feasible region, or if iterates in infeasible points are given. Modern interior-point algorithms has built-in functionality to handle these difficulties.

μ is the barrier parameter. It is a positive parameter which will be decreasing in the series of barrier problems. As μ converges to zero, the minima of φ_μ, (2.4) converges to the solution of (2.2). Note that due to the logarithmic expression, $z > 0$ must also be satisfied.

If $c(z) = 0$ in (2.4) satisfies the LICQ condition (linear independence of active constraint gradients), the solution of (2.4) with $\mu > 0$ will satisfy the first order conditions[9][7]

$$\nabla f(z) + \nabla c(z)\lambda - \mu Z^{-1} e = 0 \tag{2.6}$$

$$c(z) = 0 \tag{2.7}$$

where e is a column vector of ones; $e := [1, 1, 1, \ldots, 1]^T$ and Z is the diagonal matrix made up of the vector z; $Z := diag(z)$. λ are the Lagrange multipliers of the Lagrange function $\mathcal{L}(z, \lambda)$. Because of the extremely nonlinear behavior of the barrier function, φ_μ, solving the barrier problem directly has proven difficult. It has been fruitful to expand it to also include the dual variables of the problem. This is done by defining the dual variables y and the equation $Zy = \mu e$ to reform the primal system, (2.7) to the primal-dual system

$$\nabla f(z) + \nabla c(z)\lambda - y = 0 \tag{2.8}$$

$$c(z) = 0 \tag{2.9}$$

$$Zy - \mu e = 0 \tag{2.10}$$

This substitution and linearization reduces the nonlinearity of the barrier terms, and is also a relaxation to the KKT conditions for (2.2). Note that for μ, $z \geq 0$ and $y \geq 0$ the primal-dual system (2.10) are the KKT conditions (First Order Necessary Conditions) for the original problem, (2.2).

These equations form the base for the solution of the NLP. In general, an interior point method will use a Newton's method to solve a series of the barrier problems (2.4) via the

primal-dual system (2.10). IPOPT in particular uses a damped Newton's method. The barrier problem is solved for a fixed value of the barrier parameter μ. μ is then decreased, and continuous solving the next barrier problem from the solution of the previous one.

2.2.2 Solution of the barrier Problem

The solution to the j-th barrier problem is found by applying a Newton's method to the primal-dual system (2.11). Search directions for z, d_k^z, for λ, d_k^λ, and for y, d_k^y, are found by solving

$$\begin{bmatrix} W_k & A_k & -I \\ A_k^T & 0 & 0 \\ Y_k & 0 & Z_k \end{bmatrix} \begin{bmatrix} d_k^z \\ d_k^\lambda \\ d_k^y \end{bmatrix} = - \begin{bmatrix} \nabla f(z_k) + A_k\lambda_k - y_k \\ c(z_k) \\ Z_k Y_k e - \mu_j e \end{bmatrix} \tag{2.11}$$

A_k denotes the Jacobian of the constraints;

$$A_k := \nabla c(z_k) \tag{2.12}$$

W_k denotes the Hessian of the Lagrange function of the original problem, (2.2);

$$W_k := \nabla^2_{zz}\mathcal{L}(z_k, \lambda_k, y_k) \tag{2.13}$$

$$\mathcal{L}(z, \lambda, y) := f(z) + c(z)^T\lambda - y \tag{2.14}$$

As (2.11) is nonsymmetrical, it is not solved directly. The smaller symmetrical system

$$\begin{bmatrix} W_k + Z_k^{-1} & A_k \\ A_k^T & 0 \end{bmatrix} \begin{bmatrix} d_k^x \\ d_k^\lambda \end{bmatrix} = - \begin{bmatrix} \nabla\varphi_{\mu_j} + A_k\lambda_k \\ c(z_k) \end{bmatrix} \tag{2.15}$$

is solved instead.Symmetric matrices are represented more efficiently, and reduces computational time due to reasons discussed in chapter 2.7. It is equivalent to (2.11), and is derived by eliminating the last block. The last search direction, d_k^y is then found

$$d_k^y = \mu_j Y_k^{-1} e - y_k - Z_k^{-1} Y_k d_k^z \tag{2.16}$$

It should be noted that it in some cases is necessary to modify (2.11) some. If A_k does not have full rank, the system becomes singular, and a solution to (2.11) cannot be found. Also, for most line-search methods, the top-left block in (2.11) projected onto the null-space of A_k must be positive definit. This is to ensure some descent properties of the line-search[7]. In the IPOPT algorithm it is done by always solving

$$\begin{bmatrix} W_k + Z_k^{-1} + \delta_w(I) & A_k \\ A_k^T & -\delta_c\mathcal{I} \end{bmatrix} \begin{bmatrix} d_k^x \\ d_k^\lambda \end{bmatrix} = - \begin{bmatrix} \nabla\varphi_{\mu_j} + A_k\lambda_k \\ c(z_k) \end{bmatrix} \tag{2.17}$$

where $\delta_w, \delta_c > 0$ instead of (2.11).

When search directions are calculated from (2.17) and (2.16), the next iterates for the barrier problem are given

$$z_{k+1} := z_k + \alpha_k d_k^z \tag{2.18}$$

$$\lambda_{k+1} := \lambda_k + \alpha_k d_k^\lambda \tag{2.19}$$

$$y_{k+1} := y_k + \alpha_k^y + d_k^y \tag{2.20}$$

where the step lengths are $\alpha_k, \alpha_k^y \in (0, 1]$. Note that in some algorithms $\alpha_k = \alpha_k^y$, but in IPOPT they are allowed to have different values. This is because it is the authors of the IPOPT algorithms experience that this is more efficient [7].

2.2.3 Line-search filter for step lenght calculation

In IPOPT, the step length for the dual variable y is defined

$$\alpha_k^y := max\{\alpha \in (0, 1] \; s.t. \; y_k + \alpha d_k^y \geq (1 - \tau_j) z_k \} \tag{2.21}$$

for some $\tau_j \in (0, 1)$.

The step length for the variables z and λ, α_k are calculated uses a backtracking line-search filter method. The basic idea is to find an $\alpha_k \in (0, \alpha_k^{max}]$. Starting with the maximum step length as a candidate, the step length is decreased (backtracked) and tested until a step length giving acceptable next iterates are found. With a filter method, which is a fairly new approach, trial points are accepted if they improve the objective function *or* improve the constraint violation. Traditionally, these two measures has been combined in a single merit function. The filter method has an advantage, as it can be used to guarantee global convergence of the NLP[7].

2.2.4 Termination of Newton's method and interior point algorithm

Generally, the optimal solution to the barrier problem is found when the Newton's method is unable to find a better iterate point, and the KKT-conditions for the barrier problem are satisfied at that point. To increase the speed of the algorithm, it can be useful to stop the Newtons's method when some measurement of the of the optimality error ("Deviation from optimal point") is less than an acceptable limit. Based on the primal-dual equations (2.10), the optimality error is defined (in IPOPT) as

$$E_\mu(z, \lambda, y) := max\{\frac{||\nabla f(z) + \nabla c(z) - y||_\infty}{s_d}, \; ||c(z)||_\infty, \; \frac{||ZYe - \mu e||_\infty}{s_c}\} \tag{2.22}$$

with some scaling parameters $s_d, s_c > 1$ determined by the IPOPT algorithm.

If the approximate solution to the Barrier problem has an error less than the limit,

$$E_\mu(z_{*,j+1}, \lambda_{*,j+1}, y_{*,j+1}) \leq \kappa_\epsilon \mu_j \tag{2.23}$$

with some constant $\kappa_\epsilon > 0$, the optimal solution to the j-th barrier problem is considered to be found. The interior point algorithm will reduce the barrier parameter, such that $\mu_{j+1} < \mu_j$. Then the next barrier problem will be solved, again using the Newton's method, starting at the (approximated) optimal solution of the previous barrier problem.

This will continue, until an approximate solution, satisfying

$$E_0(z_{*,j+1}, \lambda_{*,j+1}, y_{*,j+1}) \leq \epsilon_{tol} \tag{2.24}$$

with some error tolerance ϵ_{tol} for the optimal solution of the original problem is found. Note that the test for an overall optimal solution (2.24) is (2.23) with $\mu = 0$. Generally will a larger error tolerance reduce the number of iteration, thereby reducing the computational time.

2.2.5 Interior Point algorithm summarized

A general Interior Point algorithm is outlined below:

General Line Search Interior Point Algorithm

Chose a feasible initial point (z_0, λ_0, y_0) and set $j \leftarrow 0$
repeat until the convergence test for original problem (2.24) is satisfied
  calculate the barrier parameter, μ_j
  set $k \leftarrow 0$, initalize the Newton's Method for solving the j-th Barrier Problem
  repeat until the convergence test for Barrier Problem (2.23) is satisfied
    compute search directions $(d_k^z, d_k^\lambda, d_k^y)$ using (2.17) and (2.16)
    do a (backtracking filter) line-search to find acceptable step lengths α_k, α_k^y
    calculate next barrier problem iterates $(z_{k+1}, \lambda_{k+1}, y_{k+1})$ using (2.20)
    update "inner loop" counter $k \leftarrow k + 1$
  end (repeat)
  Update "outer loop" counter $j \leftarrow j + 1$
  set start point for next barrier problem = the solution of the previous
end (repeat)
z_k is the optimal solution

Naturally, the Primal-Dual Interior Point algorithms implemented in IPOPT and elsewhere is a bit more involved, but the basics has been outlined above. For a detailed introduction to interior point algorithms i general, there are several good textbooks available [22] and [9]. For details specific to the IPOPT algorithm the implementation paper is recommended reading [7]. The IPOPT algorithm takes advantage of some clever heuristics to accelerate the algorithm, and also some advanced methods for handling infeasible start points and iterates, minimize constraint violations, scaling problem statements, etc[7].

2.3 Direct transcription

It should be noted that the IPOPT algorithm implemented later on, is a simultaneous approach, direct transcription method. That means that the system to be used as contstraints in the optimization is discretized for the whole optimization horizon, and it is assumed that the derivatives also are given for the whole horizon. As will be shown, this demands implementation of very large matrices, and the problem size is very large.

There are other approaches to optimization as well, for instance the sequential approach, where only inputs are discretized, and the system is simulated after the inputs are optimized to see if the system behaviour is as wanted. There are also multiple shooting methods, where the ODE system model is partitioned into smaller time elemtents, which are integrated separately in each elements[9]. As discussion of other methods than the implemented method with direct transcripton is beyond the scope if this thesis, it is left to the interested reader to go to the litterature [9] for a discussion of the alternatives.

2.4 Optimal path

In the context of UAVs and path planning, $f(z)$ should quantify the purpose of the flight, and the constraints should include a discretization of the UAV model. By using the UAV model as a constraint, the solution will be a path that the UAV is capable of flying.

An objective function could be as simple as the sum of the distance from the UAV to a known target, such as an iceberg, if the mission goal is to stay close to the iceberg in order to monitor it. Then the optimal solution will be the one who minimizes the UAVs distance to the iceberg. The cost function could also contain some expression of fuel consumption, in order to find an economic path as well. In fact the cost function could express several goals with different weights, in order to find the overall optimal solution.

It is also possible do a more advanced path planning, and and not only use a model for the UAV, but also model onboard surveillance equipment, sensor coverage or the uncertainty of the ice situation in a given area.

The constraints, $g(z)$ will be the UAV model itself, to ensure the optimal path is possible to follow for the UAV. Other constraints can also be added if necessary, such as limitations in speed, height, turning radiuses, start and ending positions, and limitations on flight area.

The optimization variable vector,z, includes all the UAVs states and control inputs at all times. When an optimal solution is found, one can read out the optimal position for the UAV at each time step. These positions will then be the way points in the optimal flight trajectory for the UAV. Since also the UAV's control inputs are present in z, there is an possibility to use the result of the optimization as an autopilot for the UAV, and not only path planning. Controllers (autopilots) using optimization are called Model Predictive Control.

2.5 Model Predictive Control

When calculating an optimal path with the UAV model as constraint, one has also calculated the UAV inputs necessary to follow the optimal path. This opens for using the result of the optimization as a controller for the UAV, or autopilot to use the term from the aviation industry.

Using optimization with a predictive system model is called Predictive Control, or Model Predictive Control (MPC). There are several good textbooks available, giving an introduction to MPC [18] [9]. MPC is increasingly popular due to the optimal control based on models of system behavior and also the ability handling of constraints in actuators and output using constraints.

The key concept in MPC is that the optimization is repeated, in case some unforeseen disturbance, or inaccuracy of model drives the process out of the optimal trajectory calculated in the previous iteration. The MPC should gather updated system information, solve the optimization problem again, and apply the updated solution to the process. The principle could be summarized

General Principle of MPC

Define objective, constraints and start point
repeat
  Solve optimization problem, over horizon of k time steps
  Apply/export solution for the next t_s time steps at time step $t = t_s$
  restart timer; $t = 0$
  Get updated measurements/state estimates. Update system model.
  Update objective, constraints, start point, k and t_s if necessary
end (repeat)

It is most common to used a fixed prediction horizon, k steps long. This method is known as the receding horizon or sliding window strategy. The number of steps t_s passing by before the next solution is applied depends of the time it takes to calculate the updated optimization problem, and could also be considered as a tuning factor.

Linear MPC is most popular, because solving an LP is faster and simpler than an NLP. The downside is that most processes are non-linear, and linearized models are less accurate. However, Non-linear MPC (NMPC) is increasingly popular due to development of good algorithms and increasing availability of computational power in computers and microcontrollers.

At the present, MPC is mostly used in the process industry where it is typically used to calculate optimal trajectories to lead a process to a already determined set point. The typical position of the MPC controller in the control hierarchy is shown in Figure 2.1. Most often the MPC controllers do not control the systems actuators directly. Instead it

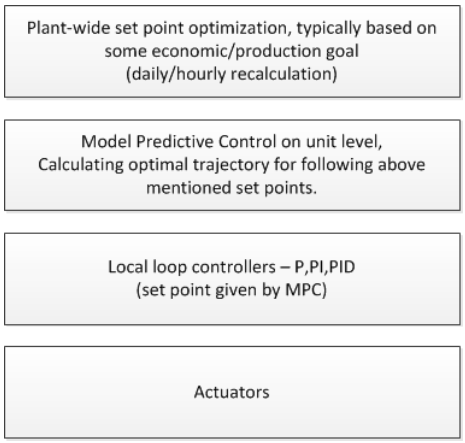


Figure 2.1: Typical control hierarchy when using Model Predictive Control

is used to give set points to standard controllers, such as P, PI and PID controllers. This is practice relates to the conservatism in the business, ("why change controllers that is known to work"), and also the fact that in the beginning, optimization algorithms were slow, safety features weres less developed and tested, and computers could be unstable. The traditional use of an MPC controller in an UAV setting is shown in Figure 2.2. Using only the calculated optimal path , and not applying the calculated optimal inputs,

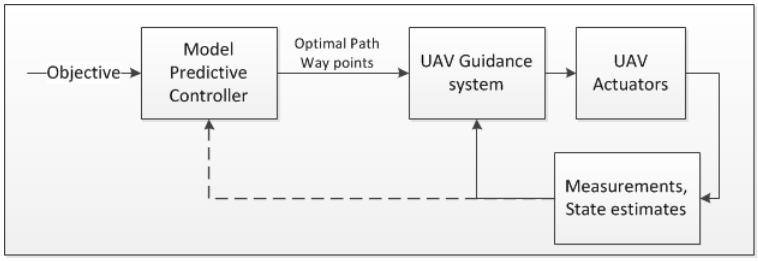


Figure 2.2: MPC controller as guidance system

is equal to using the MPC controller as a Guidance System. Path information, typically in the form of way points, are given to the UAVs control system (autopilot)[14]. Note: Only if the path is updated during a flight, it can be called MPC. If the path is only calculated once, prior to the flight, it is not MPC, but simple off line path planning.

Developments of better algorithms, research on stability and safety and faster, more powerful computers opens for applying calculated optimal inputs directly to actuators[18]. This possible future trend of MPC controller usage is shown in Figure 2.3. In an UAV

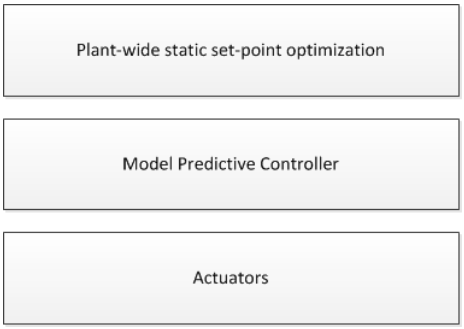


Figure 2.3: Possible future trend for controller hierarchy when using Model Predictive Control

setting, this will be equal to let the MPC controller act both as Guidance and Control system, illustrated in Figure 2.4 This latter version is implemented and evaluated in the

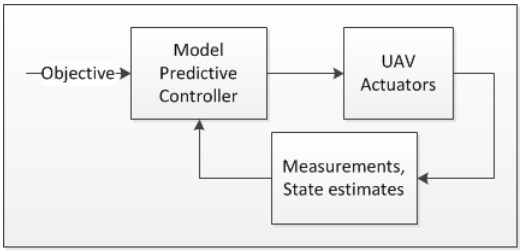


Figure 2.4: MPC controller as both guidance system and UAV controller

following chapters.

When optimization is preformed repeatidly to compute updated paths for an UAV, it is in principle an MPC controller usedonly as a guidance system. That is also called on line path planning. Therefore, the terms MPC and on line path planning are sometimes used interchangeably. It should be clear from the context wheter the MPC is ment to controll is used as Guidance, or both Guidance and Controll system

2.5.1 Safety and stability

The basis of the MPC controller is to solve an optimization problem over and over. An important tuning parameter, in addition to objective function weights, is the choice of prediction horizon, namely the choice for how many time steps ahead the process should be optimized.

It is crucial for the reliability of the MPC controller that it always finds a optimal solution. Therefor, cases such as if disturbances makes the process states drift outside constraints must be handled. It is important that the MPC controller don't produces infeasible solution. It should also be taken measures to ensure that a feasible solution, or an alternative, is found before it is time to apply the next set of inputs on actuators.

Speed-ups like "hot start", which is a technique where the solution of the previous optimization problem, or a modification of it, is applied as a start point for the next is an advantage. Handling of failing actuators and other errors should also be thought of. These advanced topics are not within the scope of this thesis, but are covered extensively in the literature[18].

2.6 Methods for discretization and simulation of ODEs

In the optimization, it is wanted to include a model of the system, which describes the systems behaviour over a given time horizon. The optimization algorithm works with discrete time variables, while physical systems typically is modeled with continous time Ordinary Differetial Equations, ODEs. Hence the need for discretization algorithms, which is equal to numerical ODE solving. There are many good textbooks on discretization/numerical integration[23].

An ODE is defined

$$\frac{\mathrm{d}\,x(t)}{\mathrm{d}\,t} = f(x(t), u(t), t) \tag{2.25}$$

$$x(0) = x^{(0)} \tag{2.26}$$

$x(t)$ being the systems states, $u(t)$ being the system inputs.

Most common are the one-step methods, either explicit

$$x^{(n+1)} = x^{(n)} + h\theta(x^{(n)}) \tag{2.27}$$

or implicit

$$x^{(n+1)} = \theta(x^{(n)}, x^{(n+1)}, t^{(n)}, t_{(n+1)}, h) \tag{2.28}$$

given a known initial value, $x^(0)$. h is the step lengt, whis is the time between to discrete points; $t_{n+1} = t_n + h$. The superscripts within paranthesis specifies the discrete time. In the implementation of an NLP in later sections a Collocation method will be use for discretization.

2.6.1 The Collocation Method

The basic idea of collocation is that the solution to an ODE, can be approximated with a set of piecewise continuous polynomials, if $f(x(t), u(t), t)$ is smooth. x or $x(t)$ denotes state variables, u or $u(t)$ denotes system inputs.

The solution to (2.26), the systems trajectory, $x(t)$ on a given time horizon is divided into S segments. All segments are T seconds long. Segment i starts node i, and ends in node $i + 1$. The function value in node i is denoted $x^{(i)}$, and $x^{(i+1)}$ in the next node. Note that the ending node of a segment is the starting node of the next segment. As of this, we see that there will be $N = S + 1$ nodes. Furthermore, the center of each segment is defined as the collocation point of the segment. The function value in the collocation point is denoted $x^{(ci)}$.

By choosing values for the system variables in each node such that the derivative of the approximating polynomial is equal to a the value of $f(x, u, t)$ in (2.26) in the collocation point, it is assumed that the nodes gives an accurate, discrete solution to (2.26). This opens for using the collocation method for numerical integration (simulation) and for discretization.

In the following, a simplified derivation av the collocation methods equations will be given. One can use several polynomials of different representations and order, as

discussed in [9]. In the following, 3rd order polynomials represented as a power series are chosen, as they are known to give a good result [16][12][10]. The derivation will be done for the first segment, starting at $t = 0$, ending in $t = T$. One will get the same result of deriving with a general segment, starting at $t = t_i$, ending in $t = t_{i+1} = t_i + T$. Note that the segment length T is the same as the general step length h in the previously mentioned one-step methods, eqs. (2.27) and (2.28).

Again, on a given segment in time, the solution to (2.26) can be approximated by the 3rd order polynomial:

$$x(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3 \tag{2.29}$$

The state values in the nodes in the beginning and end of a T seconds long segment defined by (2.29), is then:

$$x^{(0)} = x(0) = C_0 \tag{2.30}$$

$$x^{(1)} = x(T) = C_0 + C_1 T + C_2 T^2 + C_3 T^3 \tag{2.31}$$

Then the derivatives of the function values in the nodes are

$$\dot{x}^{(0)} = \frac{\mathrm{d}\,x}{\mathrm{d}\,t}|_{t=0} = C_1 + 2C_2(0) + 3C_3(0)^2 = C_1 \tag{2.32}$$

$$\dot{x}^{(1)} = \frac{\mathrm{d}\,x}{\mathrm{d}\,t}|_{t=T} = C_1 + 2C_2 T + 3C_3 T^2 \tag{2.33}$$

Note that nodes i and $i+1$ also is referred to as the left and right nodes of a segment in the literature. Definitions of nodes and their derivatives can be put on a matrix form to derive a solution for the constants C_0, C_1, C_2, C_3.

$$\begin{bmatrix} x^{(0)} \\ \dot{x}^{(0)} \\ x^{(1)} \\ \dot{x}^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & T & T^2 & T^3 \\ 0 & 1 & 2T & 3T^2 \end{bmatrix} \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} \tag{2.34}$$

Solving for the constants:

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-3}{T^2} & \frac{-2}{T} & \frac{3}{T^2} & \frac{-1}{T} \\ \frac{2}{T^3} & \frac{1}{T^2} & \frac{-2}{T^3} & \frac{1}{T^2} \end{bmatrix} \begin{bmatrix} x^{(0)} \\ \dot{x}^{(0)} \\ x^{(1)} \\ \dot{x}^{(1)} \end{bmatrix} \tag{2.35}$$

Defining a collocation point, placed in the middle of the segment, at $x(\frac{T}{2})$, inserting $S = \frac{T}{2}$ in (2.29), and using the expressions for the C's given by eq. (2.35) the Hermite interpolated state vector at the collocation point is:

$$x_c = \frac{(x^{(0)} + x^{(1)})}{2} + \frac{T}{8}(f(x^{(0)}, u^{(0)}) - f(x^{(1)}, u^{(1)})) \tag{2.36}$$

$f(x^{(0)}, u^{(0)})$ and $f(x^{(1)}, u^{(1)})$ are then eq. (2.26) evaluated in nodes to the left and right of the collocation point.

The slope of x_c, $\dot{x}_c$ is found by differentiating (2.29) with respect to t. Incerting $t = \frac{T}{2}$ along with the expressions for the C's eq. (2.35), one can derive

$$\dot{x}^{(ci)} = -\frac{3}{2T}(x^{(1)} - x^{(0)}) - \frac{(f(x^{(0)}, u^{(0)}) - f(x^{(1)}, u^{(1)}))}{4} \tag{2.37}$$

Now, if the valus in $x^{(0)}$ and $x^{(0)}$ are to satisfy the solution to the ODE, they must satisfy

$$\dot{x}^{(ci)} = \dot{x}(\frac{T}{2}) \tag{2.38}$$

where $x_{(ci)}$ is found using eq. (2.36). Note it is assumed that $f(x(\frac{T}{2}), u(\frac{T}{2}, \frac{T}{2}))$ is correctly approximated with $f(x^{(c)}, u^{(c)}, \frac{T}{2})$ such that

$$\dot{x}^{(ci)} = f(x^{(ci)}, u^{(ci)}) \tag{2.39}$$

u_{ci} is the linearly interpolated vector

$$u^{(ci)} = \frac{u^{(i+1)} - u^{(i)}}{2} \tag{2.40}$$

Eq. (2.38) is solved for $x^{(1)}$, showing the implicit one step method

$$x^{(n+1)} = x^{(n)} + \frac{3}{2T}\left(-\frac{f(x^{(n)}) - f(x^{(n+1)})}{4} - f\left(\frac{x^{(n)} + x^{(n+1)}}{2} + \frac{T}{8}(f(x^{(n)}) - f(x^{(n+1)}))\right)\right) \tag{2.41}$$

The collocation leads to a fairly involved expression, but it can luckily be reformulated for use in NLP formulations. It should also be noted that there are collocation methods where different polinnomial approximations, and more collocation points are used than in the one presented above[9].

2.6.2 Adaption of the Collocation Method for use in a NLP formulation

By expanding the collocation method, it can be left to the NLP-solver to find the values for all x^i that satisfies the solution of the ODE. A defect vector, $d^(i)$, measuring the defect (error) between the approximation and the system in the collocation point is introduced:

$$\dot{x}_{(ci)} = f(x_{(ci)}, u_{(ci)}) - d^{(i)} \tag{2.42}$$

$$d^{(i)} = f(x^{(ci)}, u^{(ci)}) - \dot{x}^{(ci)} \tag{2.43}$$

For the approximation to be accurate, the defect vector, d, should should be zero (or sufficiently small). As of this, one has to choose state and input variables in all the nodes $x^{()i}$, $u^{(i)}$, such that $d^{(i)} = 0$ for all segments. Then the approximated system trajectory made up of all $x^{(i)}$'s will be a good, discrete approximation to the solution of the time-continuous model, (2.26)

By this, there will be a defect vector for all S segment the system is deiscretized over, giving the constraint $g(z) = [d^{(0)}, d^{(1)}, \cdots, d^{(S-1)}]$

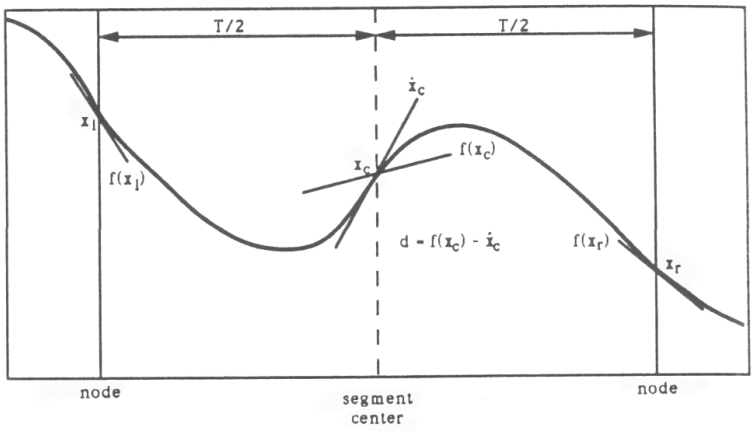


Figure 2.5: Collocation with Hermite polynomials, illustrated. Copied from [12]

2.6.3 Stability and Accuracy of the Collocation Method

It can be shown that all collocation methods are Implicit Runge Kutta methods, and that they are A-stable[9]. The A-stability is desirable, as the step length does not affect the stability properties of the solver.

A-stability means that a system modeled with stable ODE, will also be stable after beeing discretized, even if it has got system dynamics significantly faster than the step length used in the discretization [23]. However; the discretized system might become a subject to aliasing, that is that fast dynamics is lost, or downsampled such that it appears with a slower frequency. As of this, is should be notet that shorter segments give higher accuracy.

For formal definitions and a more comprehensive introduction to stability of discretisation methods (ODE solvers) and aliasing, see [23].

2.7 Matrix operations and representation

As the section describing the Interior Point algorithm shows, there will be preformed a great deal of matrix operations. Efficient algorithms for these is essential to speed up optimization algorithms. Especially methods for matrix multiplication and solving linear systems with matrices can be computationally expensive [11]. In the Interior Point Algorithm, calculating the steps with the Newton's method on the primal-dual system, eq. (2.11) needs to solved efficiently, since these steps will be calculated a great number of times[22][9]. Efficient matrix multiplication algoriths is often based on matrix decomposition and factoization.

MUMPS, the linear solver used in the implementation discussed later, solves systems on the form $Ax = b$. A must be a square matrix, wich is eiter symmetric and positive definite, symmetric or non-symmetric[1]. MUMPS uses the LU-factorization

$$A = LU \tag{2.44}$$

where L is a lower triangular matrix and U is an upper triangular matrix. If the matrix is symmetric, such as the Newton's method eq. (2.17), the factorization

$$A = LDL^T \tag{2.45}$$

where D is a block diagonal matrix. For these factorizations to work, the A matrix must be non-singular, Algorithms for matrix multiplication, factorization and linear system solving are discussed in detail in textbooks both on optimization[9][22] and those for algorithms and datastructures[22]. Most importantly, algorithms such as the one implemeted in MUMPS takes advantage of the sparsity of the matrices.

2.7.1 Compact representation of sparse matrices

A matrix with i rows and j columns contains $i \times j$ elements. In a NLP with many optimization variables, the matrices such as the Hessian becomes very large. If all matrix elements are to be stored, it will use a significant amount of memory. Therefore, alternatives to the standard representation using an $[i, j]$ array has been developed[5]. These are helpfull when the matrix is sparse; that is a significant number of the elements in the matrix are zero. That is the case for the constraint Jacobian and Lagrange Hessian of the implemented problem, discussed in chapter 3.6. The idea of the sparse formats is to only store the positions and values of the non-zero elements of the matrix, which reduces used computer data storage. This will also save time, as there will be far less read/write operations.

An example is the "Triplet Format for Sparce Matrices." This method is used by the IPOPT c++ library[8], which is used to solve an NLP in Chapter 3. The Triplet format represents the nonzeros with three vectors. Vector "iRow" holds the row indexes of the nonzeros, "iCol" holds the column indices, and "values" holds the values of the nonzeros.

This representation is best explained with an example. Consider the sparse matrix

$$\begin{bmatrix} a & 0 & b & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.46}$$

If all elements are to be stored using the standard, dense format, $4 \times 4 = 16$ elements are stored the computer memory. Many of the elements being zeros. The vactors of the triplet format will store (2.46) as shown in 2.1 The example has 4 nonzeroes, giving

row	column	value
iRow[0]=0	jCol[0]=0	values[0]= a
iRow[1]=0	jCol[1]=2	values[1]= b
iRow[2]=1	jCol[2]=1	values[2]= c
iRow[3]=2	jCol[2]=0	values[3]= d

Table 2.1: Example of triplet format for sparse matrix representation, in C++ syntax

$3 \times 4 = 12$ elements stored in computer memory when using the triple format. It is obvious that this can save a lot of memory for representing large, sparse matrices.

If it in addition is known that a given matrix is symetric, only the elements on the diagonal and the lower left corner is needed to represent it[8]. This will further reduce the number of stored elemtents. IPOPT takes advantage of this [8]. This property gives the incentive to seek to reshape matrices to become symmetric. That is done in the Interior Point algorithm presented previosly, where the Newton's method for the primal-dual system was reformulated from eq. (2.11) to eq. (2.15).

An other great advantage is that the sparse format, by defining the location of the non-zeros, helps the matrix multiplication algorithms and system solving algorithms to avoid unnecessary calculations. For instance, in a general matrix multiplication,

$$A \times B = C \tag{2.47}$$

the elemets in C are given

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \tag{2.48}$$

subscripts denoting the elements position in the matrix, and m being the number of elements in a column in A. For sparse matrises, in a great deal of these summations and multiplications either A_{ik} or B_{kj} will be a zero. The sparse matrix format helps avoiding these unnecessary computations.

Chapter 3

Implementation

In this chapter, how a program for UAV path planning using optimization was implemented is outlined. The optimization problem itself is solved with a program written in C++, which uses a Primal-Dual Filter Line-Search Interior Point method, made available through the IPOPT C++ library. IPOPT uses the following problem formulation:

$$\min_{z} f(z) \tag{3.1}$$

$$s.t. \begin{cases} g^L \leq g(z) \leq g^U \\ z^L \leq z \leq z^U \end{cases}$$

Setting up the mathematical functions needed to define and solve the NLP is mostly done via MATLAB scripts, and are then imported to the C++ program. An other MATLAB script is used to create plots to visualize the solution of the optimization afterwards. A matlab script simulating the result is also implemented, as a basis give investigate the accuracy of collocation method.

For more details on the use of IPOPT, the IPOPT tutorial[8], the IPOPT implementation paper[7] and the IPOPT wiki[6] is recommended reading. C++ source code and MATLAB scripts used in this implementation is well commented, and availible on the appendix CD.

3.1 Naming, numbering and organization of optimization Variables

In chapter 2.2, z was used to denote the optimization variables. It should be noted that the IPOPT library naming convention uses x as the optimization variables. Therefore is x used as the optimization variable in the C++ source code. In this chapter, describing the implementation, z is still used as the optimization variable, in order to easier separate optimization variables from states in the state-space models.

From the collocation, it is seen that the first segment of the discretized system introduces two nodes. For each new segment after that, a single node is added. Given that

the number of segments in the NLP is S, the number of nodes is N, and the number of optimization variables is Z, it follows that

$$N = S + 1 \tag{3.2}$$

$$Z = N \times (n + m) \tag{3.3}$$

n is the number of states in the discretized state-space model for the system, and m is the number inputs. To separate the states variables in the different nodes, $x^{(j)}$ is denotes the state vector j-th node.$x_i^{(j)}$ is the i-th state variable in the state vector of the j-th node. The same notation is used for the input vectors of the state-space models.

The optimization variables, being all states and inputs in all nodes are then organized as follows in the optimization variable vector:

$$z := [x^{(0)}, u^{(0)}, x^{(1)}, u^{(1)}, \ldots, x^{(n_{N-1})}, u^{(n_{N-1})}]^T \tag{3.4}$$

z_i denotes the i-th element of the z vector. Note that all vectors and matrixes are asumed zero-indexed, that is that the first element in a vector is element 0, and that the top left element in a matrix has position (0,0). This notation is used because C++ uses zero indexing.

3.1.1 MATLAB to C++ translation

It should be noted that variables in c++ is organized in arrays, where the first array element is read using for example " readvariable = x[0]". Tha MATLAB scripts symbolicaly calculates the mathematical function used in C++, and writes these to text files, that is included by the C++ program. However, the symbolic library is unable to work with variable names such as "x[0]" so that the corresponding variable is nemed "x0" in MATLAB and the text files. To resolve this, the #define statement in C++ is used to rename all variables in the text files, for instance "#define x0 x[0]" .

3.2 IPOPT - an open source Interior Point optimizer

IPOPT, short for Interior Point Optimizer, is a software package for large-scale non-linear optimization. It solves NLPs using the Primal Dual Filter Line-Search Method outlined in section 2.2. It is designed particularly for optimizing large, sparse NLPs, and uses the Triplet format for sparse matrices presented in chapter 2.7.1 [8].

IPOPT is written in C++ and is released as an open source code. It is available from the COIN-OR initiative under the Eclipse Public License (EPL). One is free to use IPOPT for both non-commercial and commercial purposes, and one can freely to make changes to the source code[6].

The IPOPT distribution can be used to compile a library, which can be linked to C, C++ or Fortran code. Interfaces to the AMPL modeling environment and MATLAB and R programming environments are also available. To compile this library, some third party packages for BLAS (Basic Linear Algebra), LAPACK (Linear Algebra Package) and a Symmetric Indefinite Linear Solver is needed[8].

In the following implementation, the precompiled library available from the IPOPT homepage[6] has been used. It is compiled for use with the Microsoft Visual Basic 2008 programming environment and compiler for C++. The library contains the MUMPS solver for linear algebra.

When using the POPT library, one need a object (class in c++), holding the information needed to solve the NLP. In the source code provided on the appendix cd, this object is implemented as the "Short_NLP.c" file. This object is sent to the "IpoptSolutionFactory" available through the IPOPT library, in which the problem is solved. The NLP information is implemented using several methods in the object, listed below. The NLP information comes first, then method name..

- Objective function, $f(x)$, values - *eval_f(...);*
- Objective function Jacobians values, $\nabla f(x)$ - *eval_grad_f(...);*
- Constraint functions, $g(x)$ values - *eval_g(...);*
- Constraint function bounds - *get_bounds_info(....)*
- Constraint functions jacobian, $\nabla g(x)$, values - *eval_jac_g(...);*
- Sparsity structure of constraint Jacobian - *eval_jac_g(...);*
- Number of nonzeros of constraint Jacobian - *get_NLP_info(...);*
- Upper and lower bounds for optimization variables - *get_bounds_info(...);*
- Lagrange function Hessian, $\nabla^2 f(x) - \lambda \nabla^2 g(x)$, values - *eval_h(...);*
- Lagrange function Hessian sparsity structure - *eval_h(...);*
- Number of nonzeros in Hessian of Lagrange function - *get_NLP_info(...);*
- optimization starting point - *get_starting_point(...);*

The methods takes in z_k, that is the value of all optimization variables in point k. The methods will calculate the value(s) of the corresponding function, array or matrix for the given iteration point.

Due to the use of collocation, a structure that will help implement these methods appears.

3.3 MUMPS

MUMPS (MUltifrontal Massively Parallel sparse direct Solver) is a package for solving large sparse systems of linear algebraic equations. It is compiled with, and accessed trough the IPOPT library. It is optimized to solve large sparse systems on the form $Ax = b$ with respect to the vector x

3.4 System models

System models are

...beyond the scope of this thesis. weather, fuel consumption, ice models, measurement equipment/sensor coverage models

It is 4 from the UAV model, eq: 3.5 pluss the number of uncertainty models used. m is always 2, because only the UAV model has inputs.

3.4.1 UAV model

Obviously, it should be possible for the UAV to follow the calculated optimal path. To ensure this, the state-space model for the UAV is given as a constraint to the NLP.It is discretized using the collocation method presented in chapter 2.6.1. The defect vectors from the collocation will be the equality constraints $g(x) = 0$ in the optimization problem.

In the remainder of the thesis, the following UAV model is used:

$$\begin{aligned} \dot{x}_0 &= x_2 \cos(x_3) \\ \dot{x}_1 &= x_2 \sin(x_3) \\ \dot{x}_2 &= u_0 \\ \dot{x}_3 &= \frac{g \tan(u_1)}{x_2} \end{aligned} \tag{3.5}$$

where x_1 is the position of the UAV, measured in meters north. x_1 is the position of the UAV, measured in meters east. x_2 is the speed in meters pr. second, x_3 is the heading given in radians. u_0 is the input acceleration, u_1 is the commanded banking angle.

Eq. (3.5) is a simplified model of a UAV, based on a remote controlled model airplane[10]. Only 2 of the model airplanes 6 degrees of freedom is modeled. These are the yaw (heading, x_3) and surge (speed, x_2). As the speeds and positions are given relative to some earth fixed reference point, it is assumed that the plane do not drift due to wind. As of this, it is also assumed that the speed, x_2 is the ground speed, and that it is equal to the wind speed. Furthermore, it is assumed that the UAV flies at a constant altitude, since there is no state describing the attitude of the UAV. A more accurate and detailed model would require more differential equations and variables in the model, causing a bigger NLP, which would be slower to solve. Textbooks on vessel and air plane modeling are [14] and [13].

More details on how the state-space model is implemented as a constraint follows in chapter 3.6.1.

In addition to the state-space model of the UAVs dynamics and positions, some bounds on the individual states and inputs are also given. The state-space model describes how states and inputs affect each other, while the variable bounds captures important properties such as maximum and minimum speed, turning radiuses and acceleration.

parameter	description	value
V_{min}	`Stall speed`	$11.2m/s$
V_{max}	`Maximum speed`	$26.8m/s$
a_{max}	`Maximum acceleration`	$3.0m/s^2$
a_{min}	`Maximum deceleration`	$-3.0m/s^2$
ϕ_{max}	`Maximum bank angle`	$30\,^\circ$
ϕ_{min}	`Minimum bank angle`	$-30\,^\circ$

Table 3.1: Parameter values of the given UAV constraints

These parameter bounds are :

$$\begin{aligned} V_{min} \leq x_2 \leq V_{max} \\ a_{min} \leq u_0 \leq a_{max} \\ \phi_{min} \leq u_3 \leq \phi_{max} \end{aligned} \tag{3.6}$$

These parameters are based on motor capabilities, aerodynamical features of the plane, etc. Values are given in table 3.1

The bounds are collected in two vectors corresponding to z^L and z^L in IPOPTs NLP formulation, eq. (3.1). Values for the unconstrained variables has to be present in the vectors, and are implemented as $+/- \ 2 \times 10^{-19}$, which is interpreted as $+/-$ infinity in the IPOPT algorithm.

The variable bounds for the optimization variables of the first node of the collocation is hardcoded in the *get_ variable_ bounds(...)* method in the NLP object. Loops are used to copy/expand them to cover the variables for all the nodes.

3.4.2 Information uncertainty model

In a context where an UAV is used for information gathering, it can be thought of as the UAV should minimize the uncertainty of the information on the situation in a given area. To include minimizing this uncertainty, a model for the information uncertainty could be added to the previously described UAV model. Such a model could be:

$$\begin{aligned} \dot{U}(t) &= U(\alpha - \beta(t)) + \gamma \\ \beta(t) &= b \times e^{-k(\sqrt{(T_N - x_0(t))^2 + (T_E - x_1(t))^2})} \\ P(0) &= P^0 \end{aligned} \tag{3.7}$$

α, γ, b and k being tuning parameters, T_N and T_E are the north and east position coordinates of some point of interest or target the uncertainty is related to. x_0 and x_1 are states defined in the UAV model, eq. (3.5), namely the north and east position coordinate of the UAV.

The model for the uncertainty P is inspired by the general linear system $\dot{x} = Ax$ which is stable as long as A, corresponding to $(\alpha - \beta)$, is negative (or has negative eigenvalues). Such a systems has an exponential response, and will never go below zero. This is suitable because uncertainty could be great, or none, bot not negative. $U \leq 0$. The exponential term, β is inspired by the bell curve, also called the Gauss curve, known from the normal distribution of probability. β will be zero when the UAV position (x_1, x_0) is far from the target (T_E, T_N). β will grow exponentially as the UAV approaches the target. k and b are tuning variables. $-\beta$ is plotted in figure 3.1. A bigger k will make

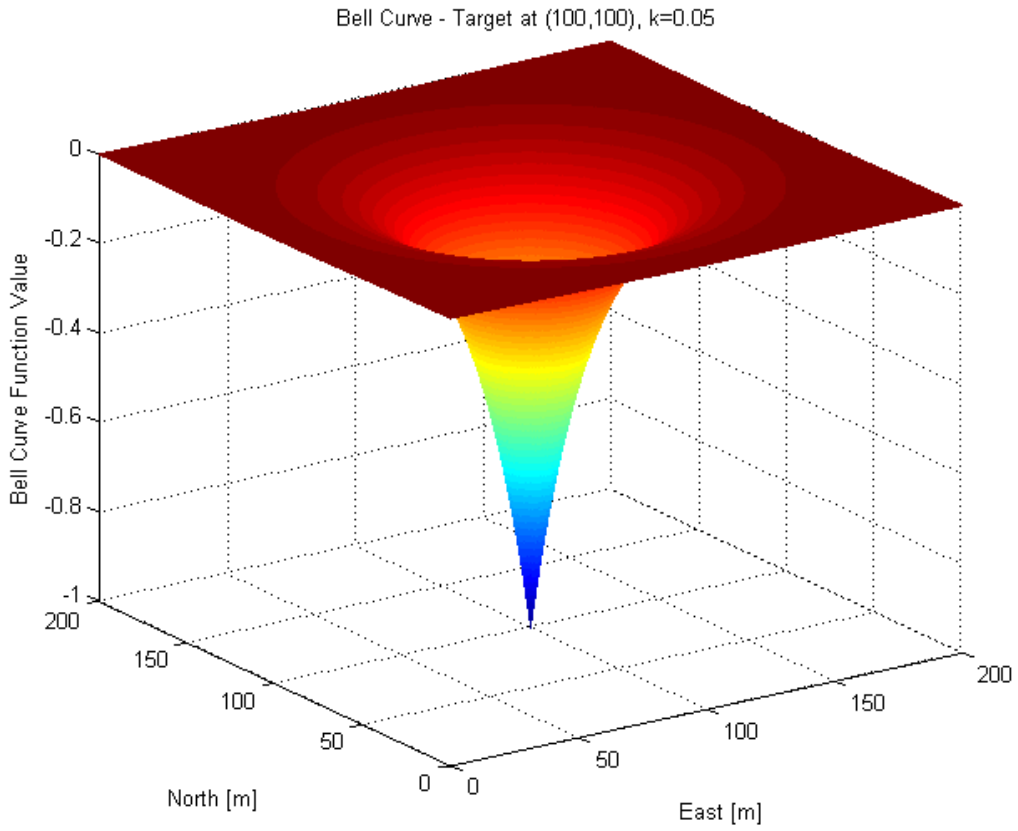


Figure 3.1: The negative Bell curve, $-f_{bell}$, with target at (100,100). Tuning variable k=0.05.

the "bell" or "funnel" shape more narrow. b decides the value of the bell function at the target. The slope of U, $-\beta U$, is clearly descending much faster directly above the target even than in positions close to the target. It seems reasonable that this shape could be exploited, so that in order for uncertainty to be reduced as much as possible, it will be favourable to go directly to the target.

α and γ are tuning parameters, ment to make the uncertainty U increase whenever

the UAV is not close to the target.

If there are several points of interest, there can be defined an uncertainty model for each of the targets. As will be shown in the following, the uncertainty model is not necessary to include in the discretized system model for all the objective functions implemented later on. But when used, the state vector for the system will be

$$x^{(j)} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ U_0 \\ \vdots \\ U_{TG-1} \end{bmatrix} \tag{3.8}$$

at the j-th node, with TG targets.

It is thought that this uncertainty model might introduce som "memory" to the system. If the uncertainty in a target can be brought to (almoast) zero when the UAV flies directly above the target, and the uncertainty does increased only slowly, or not at all, as the UAV flies away, it might be possible to find objective functions giving optimal paths that visits targets in turns. The idea is to introduce a functionality where path "remembers" where it has been, so that it is more optimal to visit a previously not visited target, than returning ta an already visited target. It is thought that this can be done by finding a parameter set where $\dot{U}$ is negative when the UAV position is close to the target positions, and most negative when the UAV is directly at the target.

As this uncertainty model not has been seen used in a path planning application before, the value og the parameters involved is a subject to experimentation to find results.

3.5 Implemented objective functions

The objective function is minimized by IPOPT, within the constraints given by the system models given in the previous section. Several different objective functions have been implemented and tested. The implemented objective functions, and the motivation for using them are presented in the following. The results of obtained by using them are given in chapter 4.

3.5.1 Minimization of quadratic distance

The perhaps simplest objective function is based on the assumption that UAV proximity to a specified target will produce useful information. Assuming this target has coordinates (T_N, T_E) specifying it's north and east positions, such an objective function on continuous-time form is:

$$f_Q(t) = \int_0^\infty (x_0(t) - T_N)^2 + (x_1(t) - T_E)^2 \, \mathrm{d}t \tag{3.9}$$

Since the UAV model is discretized over S segments, so must the objective. The integral changes to a sum, n denoting the node number:

$$f_Q = \sum_{n=1}^{N-1} (x_0^{(n)} - T_N)^2 + (x_1^{(n)} - T_E)^2 \tag{3.10}$$

Because most states the 0-th node will be given fixed values, to have a known start point for the flight, the node is not included in the objective. The optimal solution will be the one minimizing the sum of distances between the UAV and Target at each node. It is expected that the optimal behavior with such an objective function is that the UAV will seek to fly towards the Target as fast as possible, and then stay as close as possible to it during the rest of the flight. It is however difficult to foresee of this objective will encourage the UAV to repeatingly fly directly above the target, or circle around it. Note that it is assumed that the target is stationary. Also, when using this objective function, it is not necessary to include the uncertainty model in the system model, as the uncertainty states is not used in the objective.

This quadric distance minimizer can also be expanded to include multiple targets. Assuming there are TG targets subject to surveillance, all given different positions (T_N^{tg}, T_E^{tg}) a quadratic function for distance minimizing is

$$f_{Q_{TG}} = \sum_{n=0}^{N-1} \sum_{tg=0}^{TG-1} (x_0^{(n)} - T_N^{(tg)})^2 + (x_1^{(n)} - T_E^{(tg)})^2 \tag{3.11}$$

It is seen that the gradient of this function is large, when far way from the target. This should make it easy for the interior point algorithm to find a gradient necessary to find a good search direction from points far away from the target.

3.5.2 Minimization of the distance norm

Using a distance norm is based on the same assumption as the quadratic distance, namely that minimizing a measure of distance between the UAV and targets will draw the path close to, or directly above the target. For minimizing the total 2-norm distance the following function is defined:

$$f_{N_{TG}} = \sum_{n=0}^{N-1} \sum_{tg=0}^{TG-1} \sqrt{(x_0^{(n)} - T_N^{(tg)})^2 + (x_1^{(n)} - T_E^{(tg)})^2} \tag{3.12}$$

While $f_{Q_{TG}}$ has a exponential shape, $f_{N_{TG}}$ has a linear shape, which should give a gradient towards the target(s) both close to, and far way from the target. Since $f_{N_{TG}}$'s gradient is steep all the way to the target, it might help provoke the optimal path to pass closer to the targets than $f_{Q_{TG}}$, who's gradient is more flat close to the target(s).

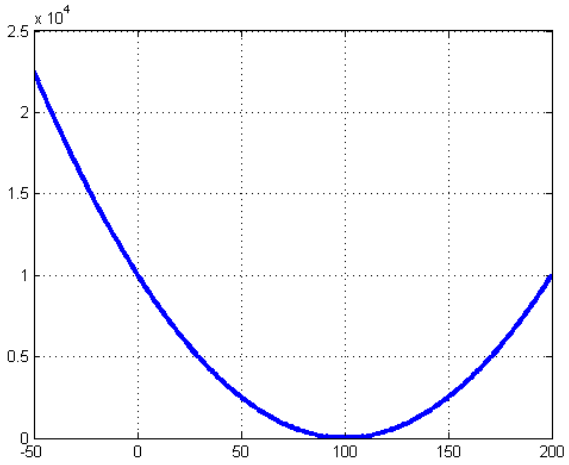


Figure 3.2: Quadratic distance minimizing with target at (100,100).

3.5.3 Minimization of distance using an exponential function

If passing directly over a target is necessary to acquire information about it, it should be reflected by the objective function. An exponential function, inspired by the "bell curve" or the Gauss curve can be used to favor paths that passes directly over the target. The the negative bell function

$$f_{bell} = -e^{-k\sqrt{(x_0-T_N)^2+(x_1-T_E)^2}} \tag{3.13}$$

is much more negative whenever the UAV position (x_1, x_0) is at the target position (T_E, T_N), than if the UAV is only close to the target. (See plot of bell curve in the previous section, figure 3.1.) It seems reasonable to belive this could be explioted to provoce IPOPT to find paths that goes directly above the target most optimal. Using this exponential function in the objective function gives

$$f(x(t)) = -\int_0^\infty e^{-k\sqrt{(x_0(t)-T_N)^2+(x_1(t)-T_E)^2}}\,\mathrm{d}t \tag{3.14}$$

in the continuous time domain. Discretized for use in IPOPT it will be

$$f_{\text{Exp Dist}}(x) = -\sum_{n=1}^{N-1} e^{-k\sqrt{(x_0^{(n)}-T_N)^2+(x_1^{(n)}-T_E)^2}} \tag{3.15}$$

A possible disadvantage with this function, is that far from the target the gradient is zero, making it difficult for the interior point algorithm to find good search directions. With this objective function, it is not necessary to include the uncertainty model in the

system model that is discretized. It can also be expanded to include multiple targets.

$$f_{E_{TG}}(x) = -\sum_{n=1}^{N-1}\sum_{tg=0}^{TG-1} e^{-k^{(tg)}\sqrt{(x_0^{(n)}-T_N^{(tg)})^2+(x_1^{(n)}-T_E^{(tg)})^2}} \tag{3.16}$$

3.5.4 Minimizing information uncertainty

With the previous presented objective function, it has not been necessary to include the uncertainty model, eq. (3.7) in the discretized system model. Only the UAV model, eq. (3.5) has been necessary. When the uncertainty model is implemented, an objective function minimizing the information uncertainty related to a given target is possible. Aan objective function for minimizing the total uncertainty across a given time horizon is

$$f_U(x) = \int_0^\infty P \,\mathrm{d}t \tag{3.17}$$

On discrete form the integral of P becomes the sum of P in all nodes over the optimization horizon.

$$f_U(x) = \sum_{n=0}^{N-1} P^{(n)} \tag{3.18}$$

For multiple targets, multiple uncertainty models are implemented in the discretized system model; one for each target. The objective function is expanded to

$$f_{U_{TG}}(x) = \sum_{n=0}^{N-1}\sum_{tg=0}^{TG-1} P^{(n)(tg)} \tag{3.19}$$

meaning that all uncertainties for all TG targets in all N nodes are summed, and will be minimzed in the optimization.

3.5.5 Additional objectives

It is also possible to combine different objectives in an objective function, and give different weights to the terms. The size of the weighting parameters, W, reflects the importance of minimizing the term.

Fuel consumption should be considered when calculating the optimal path. A good flight economy, reduces fuel costs, and makes it possible to stay in the air for a longer period of time. Assuming the fuel consumption increases with the use of acceleration, the term

$$f_{\mathrm{acc}}(x) = W_{\mathrm{acc}} \int_0^\infty u_0(t)^2 \,\mathrm{d}t \tag{3.20}$$

where W_{acc} is the weighting term and u_0 is the commanded acceleration, is added to the objective function. It will be implemented on the discrete form

$$f_{\text{acc}}(x) = W_{\text{acc}} \sum_{n=1}^{N-1} (u_0^{(n)})^2 \tag{3.21}$$

It can also be wise to add a term punishing use of the UAV rudder. It is assumed that less movement will reduce wear on hinges and actuators. Such a term is

$$f_{\text{rud}}(x) = W_{\text{rud}} \int_0^{\infty} u_1(t)^2 \, \mathrm{d}t \tag{3.22}$$

where W_{rud} is the wheighting term and u_1 is the commanded banking angle, is added to the objective function. It will be implemented on the discrete form

$$f_{\text{rud}}(x) = W_{\text{rud}} \sum_{n=1}^{N-1} (u_1^{(n)})^2 \tag{3.23}$$

3.5.6 Combined objectives in implemented functions

The term for punishing fuel consumption, eq. 3.21, and get smoother flights, eq. 3.23 are added to the objectives for distance minimizing or uncertainty minimizing. All objectives are given wheigths, reflecting their importance. The objective functions that are implemented in the path planning framework are:

$$f_{\text{Quadratic}} = W_T \times f_{Q_{TG}} + W_a \times f_a + W_r \times f_r \tag{3.24}$$
$$f_{\text{Norm}} = W_T \times f_{N_{TG}} + W_a \times f_a + W_r \times f_r \tag{3.25}$$
$$f_{\text{Exponential}} = W_T \times f_{E_{TG}} + W_a \times f_a + W_r \times f_r \tag{3.26}$$
$$f_{\text{Uncertainty}} = W_T \times f_{U_{TG}} + W_a \times f_a + W_r \times f_r \tag{3.27}$$
$$\tag{3.28}$$

The will be used in turns, testing one at a time, and for 1, 2 and 4 targets.

3.6 Structure of Implemented Functions, Constraints and Derivatives

3.6.1 Structure of the constraints

The equality constraints, $g(z) = 0$, in the optimization are all the defect vectors, coming from the collocation method used to discretize the state-space model of UAV and information uncertainty. To make $g(z) = 0$ equal to the IPOPT constraint definition $g^L \leq g(z) \leq g^U$, the constraints bounds are implemented as null-vectors; $g^L = g^U = 0$.

The constraint $g(z)$, is a column vector with the defect vectors $d^{(s_1)}$, from all the segments the system is discretized over with the collocation method. This gives the following definition:

$$g(z) = [d^{(0)}, d^{(1)}, \ldots, d^{(S-1)}]^T \tag{3.29}$$

To implement the constraints, it is taken advantage of a structure given by the collocation method. Using MATLAB, the defect vector for the first segment $d^(0)$, is calculated symbolicly. As will be shown, it is not necessary to calculate the following defect vectors. $d^{(0)}$ can be "reused", by replacing the variables in the expression. As this structure appears also when the Jacobian and Hessian of the constraints are calculated, an explanation of the structure follows here.

The first segment, s_0, stretches from the first node, n_0, to the second node, n_1. For s_0, n_0 will be the "left" node, and n_1 is the "right" node. For s_1, n_1 becomes the 'left' node while n_2 is the 'right' node. Since the collocation of both segments comes from the same ODE, the equations for $d^{(0)}$ and $d^{(1)}$ are alike, only that all optimization variable indexes are shifted with a value of $n + m$ in the latter defect vector.

$$d^0 = d(x^{(0)}, u^{(0)}, x^{(1)}, u^{(1)}) \tag{3.30}$$

$$d^1 = d(x^{(1)}, u^{(1)}, x^{(2)}, u^{(2)}) \tag{3.31}$$

$$d^s = d(x^{(s)}, u^{(s)}, x^{(s+1)}, u^{(s+1)}) \tag{3.32}$$

The calculation of $d^{(0)}$ is done by a MATLAB script. The script then translates the code for $d^{(0)}$ to a string with C++ syntax. The code for $g(z) = [d^{(0)}, d^{(1)}, \ldots, d^{(S-1)}]^T$ is then written to a text file. A loop with a search-and-replace routine replaces the variables names in $d^{(0)}$, such that $g(z)$ is written correctly. The constraint vector is made up of S defect vectors. Each defect vector has n functions, making $g(z)$ a vector with $S \times n$ elements.

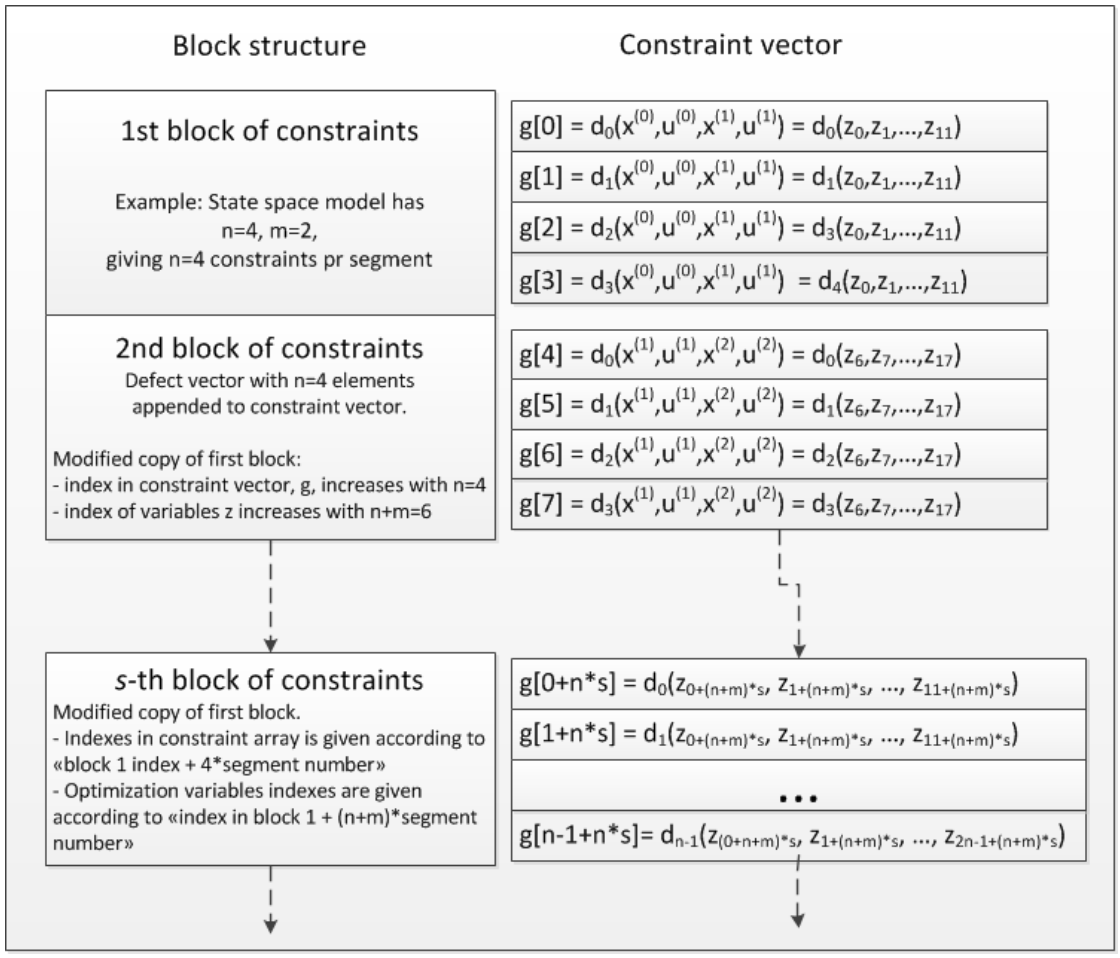
A example of this pattern is shown in figure 3.3. In the figures, the variable names follows the IPOPT naming convention, so it corresponds to the naming in the code provided in the appendix.

3.6.2 Structure of constraints Jacobian

IPOPT needs the values of the Jacobian of the Constraint vector. The constraints Jacobian is defined

$$J_g(z) = \begin{bmatrix} \frac{\partial g_1(z)}{\partial z_1} & \frac{\partial g_1(z)}{\partial z_2} & \cdots & \frac{\partial g_1(z)}{\partial z_{N\times(n+m)}} \\ \frac{\partial g_2(z)}{\partial z_1} & \frac{\partial g_2(z)}{\partial z_2} & \cdots & \frac{\partial g_2(z)}{\partial z_{N\times(n+m)}} \\ \vdots & \vdots & \ddots & vdots \\ \frac{\partial g_{S\times(n+m)}(z)}{\partial z_1} & \frac{\partial g_{S\times(n+m)}(z)}{\partial z_2} & \cdots & \frac{\partial g_{S\times n+}(z)}{\partial z_{N\times(n+m)}} \end{bmatrix} \tag{3.33}$$

Figure 3.3: Example of constraint vector structure



Again, there is a structure that can be exploited when implementing this. Using MATLABs ability to differentiate exact using its Symbolic Toolbox, only the Jacobian of the first segments constraints, $\nabla_z d^{(0)} = \nabla_z [g_0, g_1, \ldots, g_{n-1}]^T$ needs to be calculated. $g_i(z)$ is the i-th element of the g vector.

$\nabla_z d^{(0)} = J_{d^{(0)}}$ is the top left block of $J_g(z)$. $J_g(z)$ will be a diagonal matrix, with blocks, $\nabla_z d^{(0)}$, $\nabla_z d^{(1)}$,$\nabla_z d^{(2)}$ and so on along its diagonal.

All defect vectors are calculated from the same ODE, which makes all blocks $J_{d^{(s)}}$ equal, only with variables "shifted" one node to the right for each segment. That means that the indexes of the optimization variables in the expressions in the block increases with $(n+m)$ for each block. In the s-th block, $s \times (n+m)$ is added to the index of the variables of the first block.

The index of the top left corner of the blocks changes with n positions for the column index, and $(n+m)$ positions for the row index. With the top left indexes $[0,0]$ for the

0th block, the top left corner of the s-th Jacobian block has index

$$I^{row}_{Jb(s)} = I^{row}_{Jb(0)} + s \times n \tag{3.34}$$
$$I^{col}_{Jb(s)} = I^{col}_{Jb(0)} + s \times (n+m) \tag{3.35}$$

Figure 3.4 shows this structure.

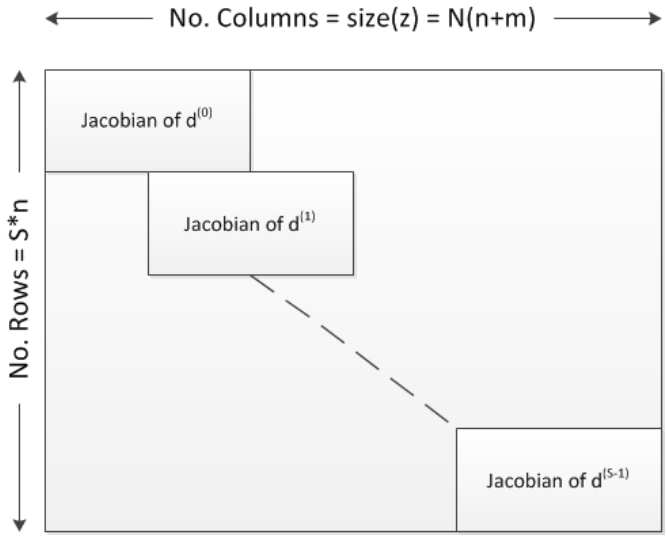


Figure 3.4: Block structure of Constraints Jacobian

Similar to the method used setting up the constraints, the exact expression for $J_{d^{(0)}}$ is calculated using a matlab script. The code for the block is translated to C++ syntax, and written to a text file. Using a loop with a search-and-replace function, all variable indexes from the 0th block is increased with the term $s \times (n+m)$, and position indexes updated according to eq. (3.35).

The matrix is implemented in the previously discussed sparsity format.

3.6.3 Structure of Lagrange function Hessian

Also the Hessian of the Lagrange functions, W, is needed to solve the problem, as it appears in the primal-dual system, eq. 2.15 used to find search directions for the Interior Point algorithm. It is implemented with a scaling parameter σ_f. This is due to IPOPTs problem scaling ability, and the second derivatives checker. As of this, the Lagrange Hessian in point z_k is defined

$$W_k = \sigma_f \nabla^2_{zz} f(z_k) + \sum_{i=0}^{(S-1)\times n} \lambda^{(i)} \nabla^2_{zz} g_i(z_k) \tag{3.36}$$

$g_i(z)$ is the i-th element of $g(z)$. Note that the first n elements in $g(z)$ are the elements in $d^{(0)}$. Written as a matrix, the Hessian is defined

$$W = \begin{bmatrix} \frac{\partial^2 L(z)}{\partial z_1^2} & \frac{\partial^2 L(z)}{\partial z_1 \partial z_2} & \cdots & \frac{\partial^2 L(z)}{\partial z_1 \partial z_{N\times(n+m)}} \\ \frac{\partial^2 L(z)}{\partial z_2 \partial z_1} & \frac{\partial^2 L(z)}{\partial z_2^2} & \cdots & \frac{\partial^2 L(z)}{\partial z_2 \partial z_{N\times(n+m)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L(z)}{\partial z_{N\times(n+m)} \partial z_1} & \frac{\partial^2 L(z)}{\partial z_{N\times(n+m)} \partial z_2} & \cdots & \frac{\partial^2 L(z)}{\partial z_{N\times(n+m)}^2} \end{bmatrix} \tag{3.37}$$

A block structure similar to the structure in the Jacobian appears. The key is to see that the first n elements in $g(z)$ are actually the elements in $d^{(0)}$ which is the defect vector of the fist segment. These n elements, which are functions, appears also in the second term of W_k, twice differentiated. They only variables from the two first nodes. The next n elements, g_n to g_{2n-1} comes from $d^{(1)}$, which contains the same functions as $d^{(0)}$, only with variables from the second and third node.

Because of this, it is only necessary to differentiate $d^{(0)}$ symbolicaly to find the exact Hessians. The second derivatives of the next defect vecors is found by simply replacing variables in the expressions for $\nabla^2_{zz} d^{(0)}$.

W is a matrix with blocks on its diagonal, each block being the second derivatives of a defect vector from the collocation. There will be S Hessian blocks. The definition for the sth Hessian block at point z_k is

$$W_k^{(s)} = \sigma_f \nabla^2_{zz} f(z_k) + \sum_{i=s}^{s+(n-1)} \lambda^{(i)} \nabla^2_{zz} g_i(z_k) \tag{3.38}$$

Defining a vector holding the Lagrange multipliers related to the n constraints (the defect vector) of a segment:

$$\lambda_{\text{vec}}^{(s)} = [\lambda_s, \lambda_{s+1}, \ldots, \lambda_{s+n-1}] \tag{3.39}$$

the expression for the s-th Hessian block can be rewritten to

$$W_k^{(s)} = \sigma_f \nabla^2_{zz} f(z_k) + \lambda_{\text{vec}}^{(s)} \nabla_z (\nabla_z d^{(s)}(z_k)) \tag{3.40}$$

This time, the setting up the Jacobian with the block structure is a bit more involved, as the blocks will overlap each other. This is because both segment s=0, and segment s=1 contains variables from node 1. Remember from the collocation that the "right" node of a segment becomes the "left" of the next.

The block structure is shown in Figure 3.5.

Each Hessina block, $W^{(s)}$ has dimention $(2 \times (n+m)\,, 2 \times (n+m))$. Because the lower right quarter of a block is overlapped by the upper left quarter of the next, the index of the top left element in the s-th block is given (assuming zero-indexing)

$$I^{row}_{W^{(s)}} = I^{col}_{W^{(s)}} = s \times (n+m) \tag{3.41}$$

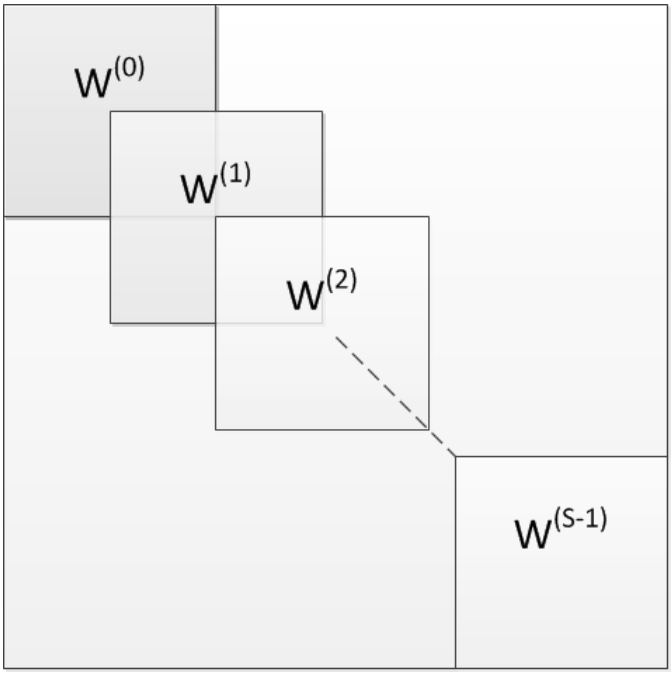


Figure 3.5: Block structure of Lagrange function Hessian

In the implementation, the objective function Hessian and the constraint Hessian for the first block are calculated separately. They are then added to create the Lagrange Function Hessian for the first block. It is found using a MATLAB script, using symbolic differnetiation. THe expression for the block is translated to C++ syntax and written to a a text file with C++ syntax. This generalization where the first block is reused to find all blocks are done in the C++ program. If the whole Hessian Matrix was written to a text file, it could be more than 10 MB for this implementation, and then it would take hours to compile the c++ program, when such a big file was to be included. Each block turns out to be quite dense, but W_k is sparse, as it's elements are on the block diagonal. Due to the mentioned overlapping of blocks, the indexes of where in the matrix functions in a block is appended to functions from the previous block has to be known.

The Hessian matrix is implemented using the sparse matrix format. It is also noted that the Hessian matrix is symmetric. IPOPT knows this, and therefore only the lower left triangle, including the diagonal is written.

3.7 Implementation of MPC controller/On line path planner

The MPC controller was implemented by expanding the framework for off line path planning with a for loop. In this for loop, the object holding the NLP information was sent to the method for solving the NLP repeatedly. A very useful feature in the

IPOPT framework is that the object with the NLP information has a method called *finalize_solution(...)* which is called after an optimal solution to the NLP is found. In this method, all functionality for printing results are put, together with the functionality for updating the NLP information, making it ready for the next MPC iteration.

3.7.1 Path start point

The optimal UAV starts at node 0. Therefore is most variables in node 0 given bounds, where the upper bound is identical to the lower. These bounds define the start position of the uav path, the start heading, the start speed and the start uncertainties. Only inputs are left with the variable bounds given by the UAV model. This becase the inputs in node 0 will affect the states in node 1. In the MPC controller the start point for the next optimization is set in the *finalize_solution(...)* method.

3.7.2 Scenario 1 - Updated target info

In the first test scenario for online path planning/MPC, it is assumed that the target information is given by an outside source. A new path is planned when new target infomation is recieved. In each optimization it will be assumed that the target is stationary, such that the previously discussed objective functions and uncertainty model is used.

The point is to see how the flewn UAV path turns out when the target information is frequently updated, to see if this principle can be used for tracking of moving ice bergs.

New target information is simulated by letting the *finalize_solution(...)* method update the target positins.

3.7.3 Scenario 2 - Wind disturbance

In the second scenario, it is assumed that the UAV is subject to a wind disturbance. The disturbance drives the UAV of the first calculated optimal path, and a new path is calculated, so that the UAV will have a new optimal path to lead it to the target ice bergs. The wind disturbance is simulated in the *finalize_solution(...)* method, by altering the UAV position variables.

3.8 Implementation of simulations

The optimization of the optimal path will also calculate the inputs necessary for flying it. In order to investigate the accuracy of the collocation method, a RK4 method has been written in MATLAB for comparison. The RK4-method takes the calculated UAV acceleration and bank angle inputs, and simulates a flight. The same state-space-model for the UAV is used in the collocation and the RK4 simulation. AS the RK4 method will is implemented with the possibility to use much shorter step lengths than the T used in the collocation, it will use a first-order-hold (FOH) on the input between nodes. This is because the collocation also uses FOH on inputs between nodes.

3.9 Generating initial points for IPOPT

Because it was not found time to code a simulator using collocation as an implicit ODE solver, deined in eq. 2.41, start are generated with the easier-to-implement RK4 method. Given a defined start point for the path (the state variables in node 0, $x^{(0)}$), the path flewn with this start point, with only zeros as inputs, $u^{(i)} = 0$, is used as an initial point for the IPOPT algorithm.

As previously discussed , collocation are implicit RK methods. The used RK4 method is explicit, and of a different order. As of this, the given start point is slightly infeasible. IPOPT has built-in-functionality for handeling this.

3.9.1 Hot Start

In the MPC controller used in the wind disturbance scenario, a "hot start" functionality is implemented. This is a functionality where the optimal path found by the optimization is used as the initial point in the next MPC iteration. It is assumed that the two optimal paths will be quite similar, so that fewer iterations and less computational time is needed to solve the next optimization.

Chapter 4

Results

In this section the results from using the implemented path planning framework will be presented and discussed.

4.1 Software and hardware

The C++ programs with the optimization algorithm implemented are written and compiled using Microsoft Visual Studio 2008. It is run on a Dell Optiplex 780 PC, with an Intel Core2 Duo 3.00 GHz CPU, and 4.00 GB RAM. The PC runs the Microsoft Windows 7 32-bit Enterprise Service Pack 1 operating system. MATLAB R2010b is used to run the scripts setting up mathematical functions and plots. It should also be noted that when solving the different optimization problems, the default values in IPOPT for error and infeasibility tolerances was used.

4.2 Note to plots

In the following, multiple plots of the results are presented. Since the IPOPT optimizes state variables of a discrete-time model, all states are plotted vs time using zero-order-hold, or a "stair plot" as it is called in MATLAB. The exception being the plots of UAV paths, where the east positions $x_0^{(n)}$, are plotted vs north positions $x_1^{(n)}$.

Common for all plots is that a circle on the line corresponds to a node (defined in the collocation). Remembering that theres a node in each end of a segment, and a segment is T segments long, the circles in the plots are placed T seconds apart.

4.3 Test cases

4 different objective functions are tested, all in different versions depending on the number of targets defined. The imagined test scenario is that the targets specified either in the objective functions or by the uncertainty functions represents ice bergs with a known position. Even though no choice has been made of what kind of surveillance equipment is

installed onboard the UAV, it is assumed that flying directly above the target(s) as many times as possible is beneficial. When testing different weights and tuning parameters, the goal has been to fly above each target at least once.

When introducing multiple targets, each target could also be thought of as representing the center coordinate of a set of frames in a search grid. In a more general surveillance mission it could be benefiting to define a search grid, and have the UAV visiting the centers of each grid at least once in order to investigate the ice condition in that general area. When tuning optimization with many targets, weights and parameters making the path pass directly above the targets (grid frame centers) are wanted.

4.4 Optimal Path with minimization of distance to single target

In this section the results from the first implemented optimal path planner is presented. The used objective function is the simple quadratic distance minimizer (3.11), with added punishing terms on acceleration and rudder inputs.

$$f_{min} = Wt \times f_{Q_1} + W_a \times f_a + W_r \times f_r \tag{4.1}$$

The uncertainty model is not included in the discretized system model, as its states is not used in the objective. Only the UAV model, eq. [?] is used in the discretized system, making up the constraints. The targets position is $(100, 100)$, and the wheights in the optimization are The size of the wheighting term reflecs the individual terms

Wt	10.0
Wr	5.0
Wa	120.0

Table 4.1: Objective function wheights - Quadratic Distance Minimizing, single target

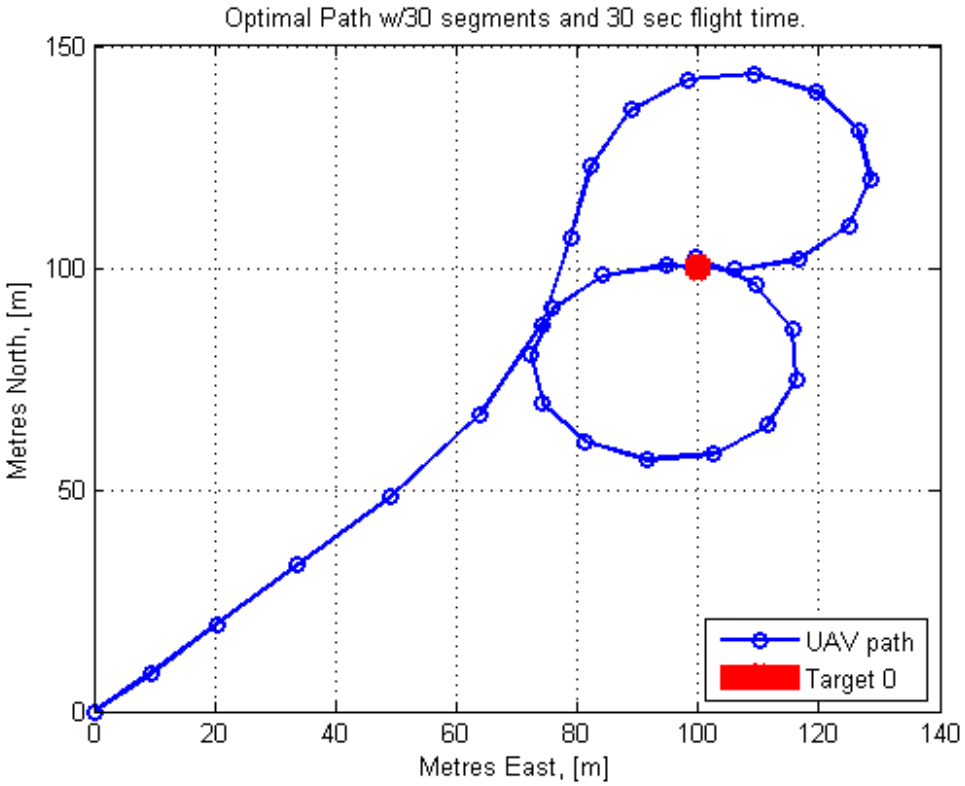
importance. It should however be noted that the sum of accelertion and rudder inputs are very low compared to the sum of distances between UAV and target. Therefore might Wa seem very big compared to Wt, even though the UAV being close to the target is most important. Any lower values of Wa resulted in rapid oscillations in the acceleration input, where the input was alternating between max and minimum at every other time step for parts of the flight. The punishment for using that input was increased until the inputs became smoother, which is favourable as it will decrease wear end tear of the system.

It took the implemented framework 0.733 seconds to find the optimal path. The initial point of the optimization was a straight path with minimum speed, in direction 1 (radian). That is roughly a north-east direction in the plots were the x-axis is the east positions, $x_1^{(i)}$ and the y-axis is the north positions $x_0^{(i)}$.

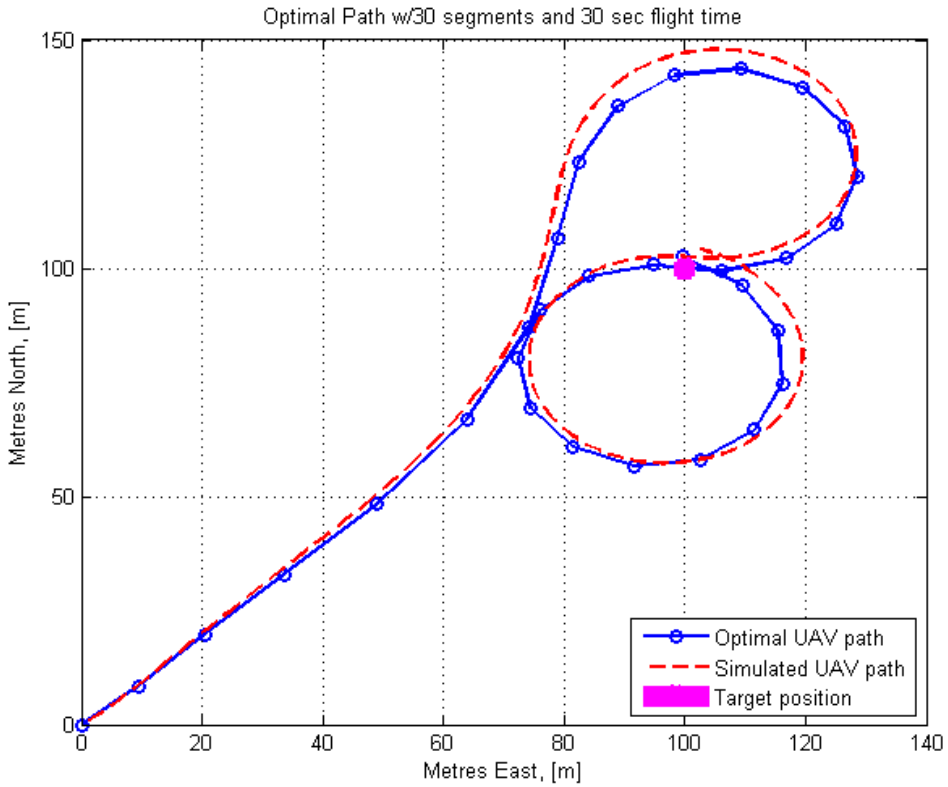
With these wheights, the optimal path is seen in Figure 4.1a. The optimal path resembles a "figure 8" shape, and passes over the target twice. It is seen that is more optimal for the flight to end directly on top of the target, than flying directly to it in the beginning. The path actually loops around the target at first, before it passes above it after about 17,5 seconds, and then again after 30 seconds.

4.4.1 Feasibility of path

From the UAV model it is seen that the UAV speed is affecting the change in heading. Slower speed enables tighter turns. It is seen in from fig. 4.2b that the UAV at first speeds up to fast minimize the distance to the target in the beginning of the flight. Then the UAV slows down in order to perform turns as tight as possible, in order to stay close

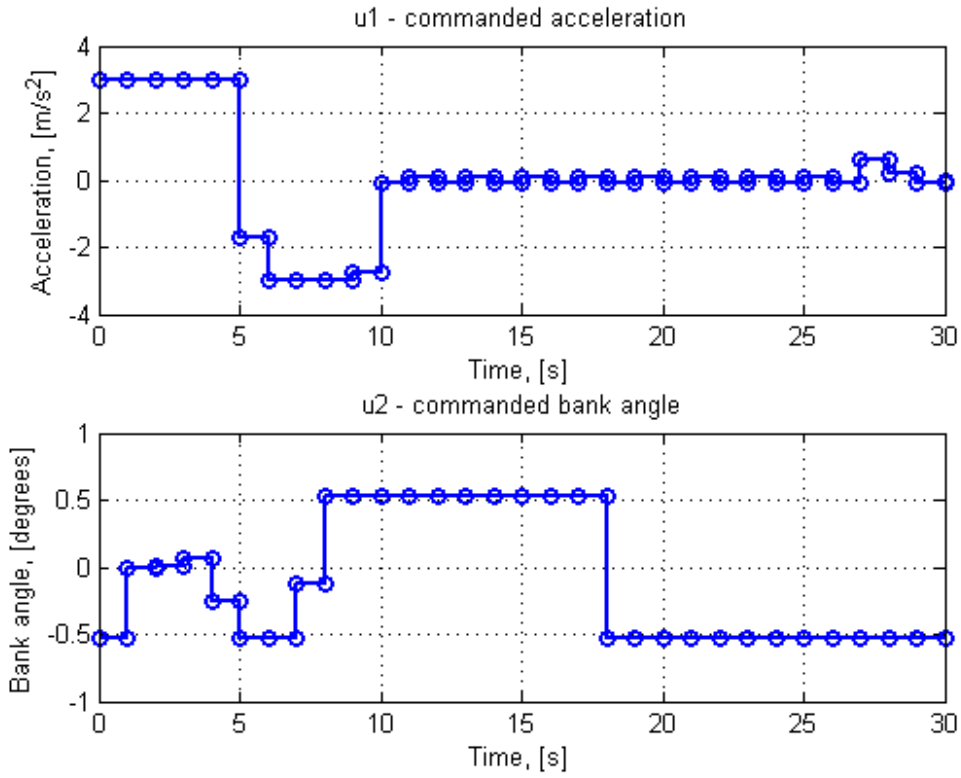


(a) 30 sec. optimal Path with quadratic disatance minimizing to a single target

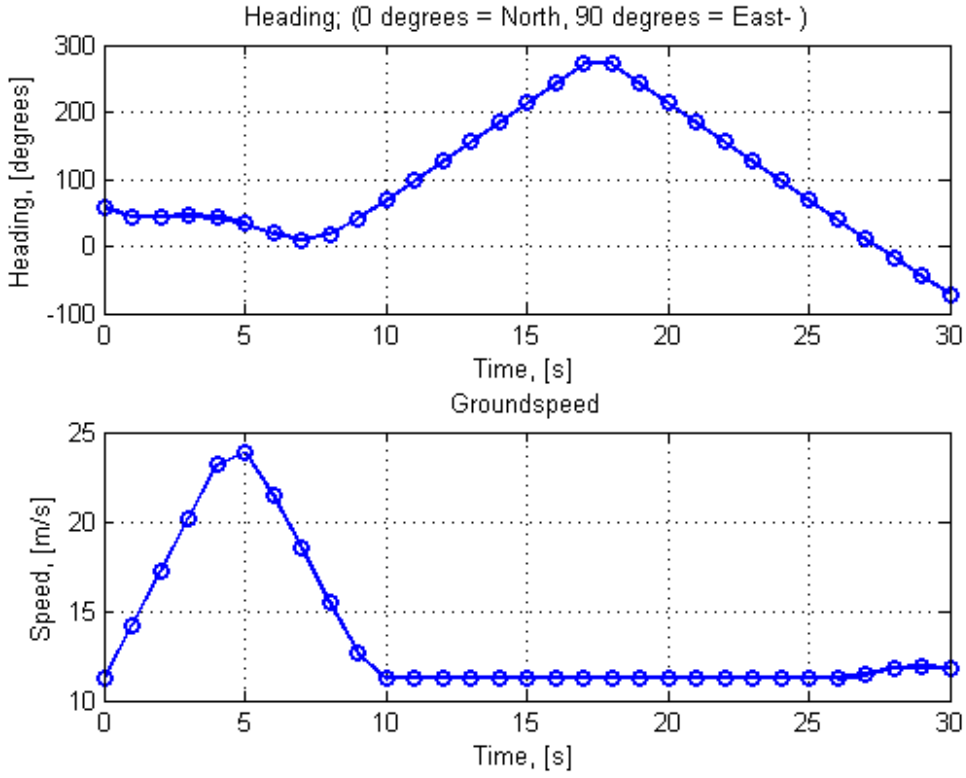


(b) Optimal path and path simulated with RK4 using inputs from optimal path

Figure 4.1: Optimal paths, quadratic distance minimizing and single target



(a) Optimal UAV inputs for 30 sec flight with quadratic distance minimizing to a single target



(b) UAV speed and heading during 30 sec flight with quadratic disatance minimizing to a single target

Figure 4.2: Other data, quadratic distance minimizing and single target

to the target. For most of the flight the speed is $11.2m/s$ which is the lowest allowed speed.

One can also see that graphs for acceleration in fig. 4.2a and speed in fig. 4.2b corresponds well. The UAV is accelerating the first 5 seconds, and in the same time the speed is increasing. Both graphs for commanded bank angle and the UAV heading corresponds well to what is seen in the ploted path. First turning one way, then the other is seen in the "figure 8" shape of the path.

The graphs of inputs are consistent with the plotted path and the plots of UAV speed and direction. This indicates that the UAV model has been implemented correctly. All variables are within the variable bouds given with the UAV model. It seems reasonable that the implementation results in paths the UAV is capable of flying.

To further very that the result is correct, the calculated inputs were used as inputs in a simulation with an RK4, with the same UAV model used in the collocation. The simulated path as ploted along with the optimal path in figure 4.2b. The two paths are quite similar. Some difference is expected as the collocation method is implcit, while the RK4 method is explicit, nad they ar of different order. In the RK4 method, the step length is also $T/10$ which should make it a more accurate simulation. Accuracy of the discretization given by the cvollocation method is discussed further later on. The likeness of the two paths further supports that the framework is implemented correctly, and that the paths are optimal and feasibe, so they can be followd by the UAV.

4.4.2 Usefullness of distance minimizing in target surveillance

It is also clear that using an objective function based on minimizing the distance between a UAV and a target can give paths that goes directly (or very close) to the target. It should be noted that with different wheights in the objective, the optimal path sometimes goes in a circle around the target, with the target in the centre of the circle. In those cases the target is not flewn across.

It was also seen that the results when minimizing the distance norm, or the exponential distance, the results were so similar to the quadratic distance minimizing presented here, that it was not found necessary to show plots of these when only a single target is used.

4.5 Step lenghts and accuracy of the Collocation Method

To investigate how accurate the collocation method discretizes, the inputs calculated for the optimal path, are used as inputs in a simulation. For the simulations, the RK4 method presented in the implementation chapter, chapter 3, is used with step lenght $h = 0.1 \times T$. As the RK4 method is known to give accurate approximation of ODE's when using short step lengths, it is concidered to represent the "real world" behaviour of an UAV using the calculated inputs.

First, 30 segments long optimization is done with a short step length, $T = 0.5$ seconds. This gives step length $h = 0.05$ in the RK4 method. The result of this optimization is

shown in fig. 4.3a. Beneath, in fig. 4.3a, the optimal path and simulation is calculated with $T = 2.0seconds$ and $h = 0.2$ is shown.

From the figure it is seen that the simulated path runs closer to the calculated optimal path. With $T = 0.5$ the deviation is approximatly 1-2 meters at the final node. It is seen that for $T = 2$ the difference between the optimal path and the simulated path is bigger. The difference does also grow bigger during the flight. The shape of the optimal path with the longer step length is more complicated, as the UAV has time to circle over the target, which contributes to the big difference of accuracy at the last nodes of the paths. With step the longer step length, the biggest deviation is approximetly 20 meters.

4.5.1 Segment length vs path lenght

Figure 4.3 also illustrates an other important aspect when choosing step lengths. For shorter step lengths, more segments are needed to calculate a path of the same length. More segments gives more nodes, meaning more variables in the optimization problem. Hence there is a trade-off between accuracy and problem size, as bigger optimization problem is typically slower to solve.

4.5.2 Solution times with varying T

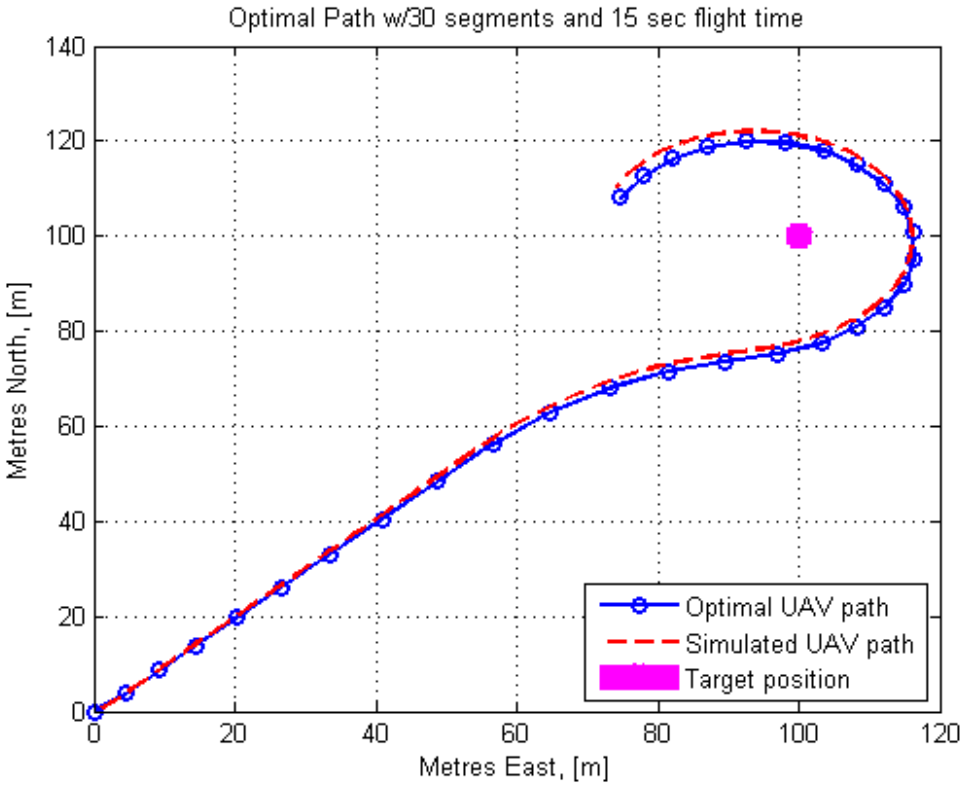
It was also noted that the step lengths are influencing the computational time of the IPOPT algortihm. Computational times and number of iterations in the algorithm for different values of T is avalible in table 4.2. Table 4.2 shows a clear trend that longer

Segments, S	Segment lenght, T	Iterations	CPU Time (sec)
30	0.5	145	0.421
30	1.0	193	0.733
30	2.0	266	0.857
30	5.0	370	1.325

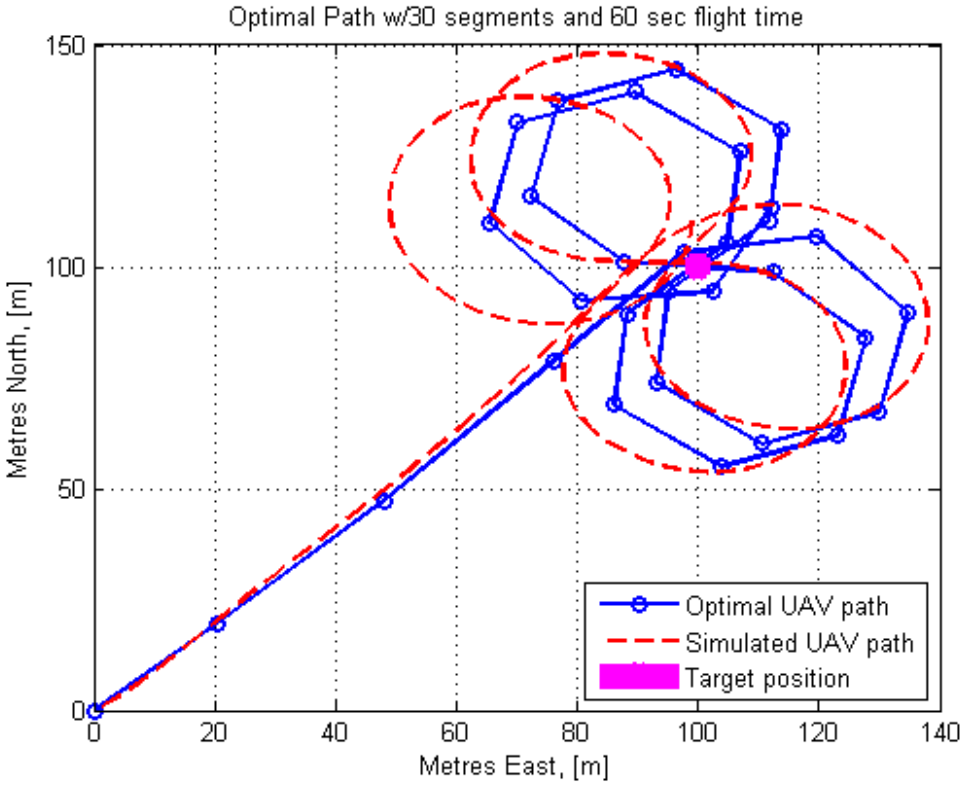
Table 4.2: Computational times for different Segment lengts, 30 segments

step lenghts gives longer computational times. Having showed that shorter steps gives a more accurate, this seem to make it easier for the algorithm to find god Newton steps. However, the picture is a little more complex. Becuse of the RK4 method used to generate the start point of the algorithm, longer steps will gives more infeasible start point, which could make it slower to return to feasible points. Longer steps will also give more complex UAV dynamics, seen from 4.3. With longer step lengts, there will be time for the UAV to perform more and longer turns, in which it could be difficoult to find good Newton steps for next iterates.

Wheights will also influence computational times, at is influences what search directions, and step lengths that are accepted before finding the optimal path.



(a) Optimal and simulated UAV path with shorter step lenght; T=0.5



(b) Optimal and simulated UAV path with longer step length; T=2.0

Figure 4.3: Effects of step length

4.6 Exact vs approximated Hessian

The IPOPT library comes with an option of using an approximation instead of the exact Hessian. The Hessian is then approximated with the Quasi Newton BFGS method. Since the expressions in the Hessian can be quite complexe, and it proved time consuming to implement it correctly, it is of interest to see what difference using an exact Hessian makes. In table 4.3, computational times are compared for optimizations, using the quadratic distance minimization objective on 1 and 4 targets.

S	T	Targets	Iterations	CPU Time (sec)	Comments
60	1.0	1	145	0.847	BFGS, acceptable solution
60	1.0	1	307	1.567	Exact Hessian,Optimal Solution
60	1.0	4	644	4.103	BFGS, acceptable solution
60	1.0	4	306	1.530	Exact Hessian, optimal solution

Table 4.3: CPU times for quadratic distance minimizing. Exact and approx Hessians

IPOPT detects if the algoithm is unable to decrease the infeasibility sufficently to find the optimal solution. If the error is below a given value (the aceptable error), the iterations stops, and the algorithm calls it an acceptable solution[8].

For the simplest optimization; distance minimizing to a single target, using the BFGS approximation gave a little shorter CPU time. It was however often unable to find the optimal solution, and terminated at only acceptable points instead. Such an example is given in table 4.3 For more complex optimization problem, such as multiple targets, or unsing the uncertainty model, an exact Hessian was necessary to find the optimal point, and it was also able to find the optimal point with much fever iterations than the BFGS. There were also clear visible differences between the optimal and the acceptable soultions. In the acceptable solutions, the paths tended to go a bit to the side of the targets, instead of directy across them.

4.7 Stopping criterions

That stoping at acceptable points might be faster than finding optimal solution does also serve as a remainder that size of error and infeasibility tolerances, determining the termination of solving the individual barrier problems or the overall NLP, will affect computational times. If less accuracy is needed, larger errors can be tolerated, leading to fewer iterations and less CPU time.

A possible problem arises if one is to increase the infeacibility tolreance. The optimization variables includes both UAV position, and it's heading. For the UAV an accuracy of +/- 1 meter might be acceptable, but +/- 1 radians is very much. IPOPT uses the max norm when checking for constraint infeacibility, so one must set the infeacability tolerence according to the most sensitive state values.

No choise has been made on what an acceptable level of accuracy is for the cases in this thesis. Therefore has the defalt IPOPT values for termination been used.

4.8 Optimal path with multiple stationary targets and distance minimizing

In this section, the three objective functions for minimizing distanze to multiple targets are tested. Since the slope of the objective functions have different shapes close to the targets, it will be of interest to see how different the calculculated optimal paths will be.

Using multiple targets is interesting for several reasons. Each target could represent an object, such as ice berg, needed to be investigated in order to establish it's exact size, position and direction. An other use is to let the targets be either centers or crnesr of a search grid quadrant, and see if this will produce a path making the UAV fly through all search grids. This could be useful in surveillance missions where theres no known object to investigate, and a more general search through a defined area is wanted. It should be noted that when tuning parameters, the objective in mind has been to fly diractly above the targets.

All minimizing objectives are combined with objectives for minimizing controll efforts, for en economic flight, giving the following used objective functions.

$$f_{\text{Quadratic}} = W_T \times f_{Q_{TG}} + W_a \times f_a + W_r \times f_r \tag{4.2}$$

$$f_{\text{Norm}} = W_T \times f_{N_{TG}} + W_a \times f_a + W_r \times f_r \tag{4.3}$$

$$f_{\text{Exponential}} = W_T \times f_{E_{TG}} + W_a \times f_a + W_r \times f_r \tag{4.4}$$

$f_{Q_{TG}}$, $f_{N_{TG}}$ and $f_{E_{TG}}$ were defined in Eqs. (3.11), (3.12) and (3.16).

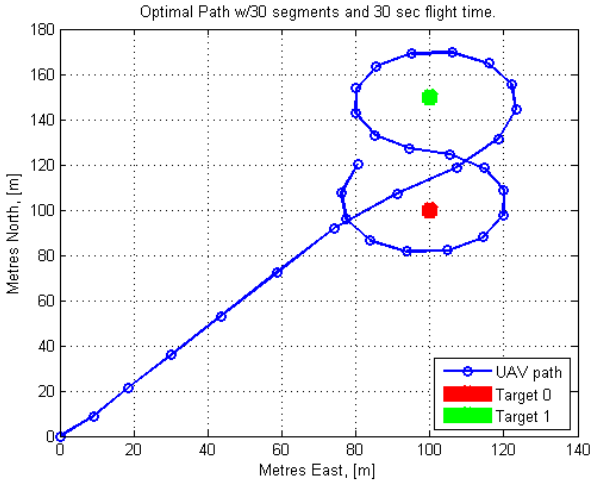
4.8.1 Surveillance of 2 targets

The results of the optimization with 2 targets are shovn in fig. 4.4.

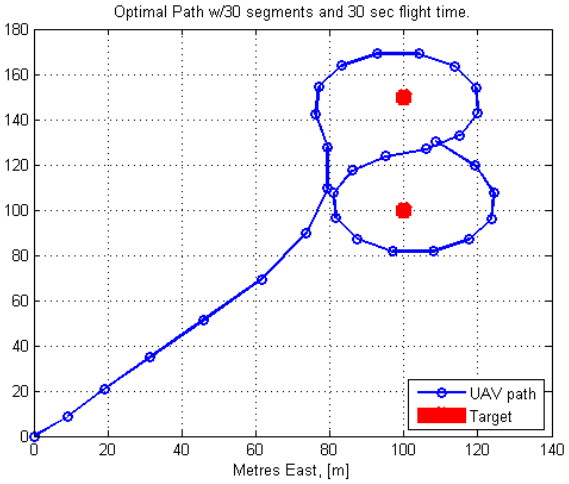
It is interesting to see that neither of the used objective functions made it benifitial to fly above the target in the beginning of the flight. This comes from the use of suming up of all distances, so that it's not necessary to fly the shortest possible way to a target first. The quadratic distance minimization in fig. 4.4a and the norm distance minimizer, fig. 4.4b both finds a "figure 8" shape optimal, but flewn in opposite directions. In both cases the UAV circles around the targets, insted of passing directly above it. In this case it is clear that the objective function is not redused enough when flying directly above it, than circle around it.

The exponenial objective in fig. 4.4c is significantly different. It is seen that it has found it optimal to first fly towards the second target, and then fly directly above the southenrmost target twice. It seems that the idea of using an objective inspired by the inverted bell curve is indeed making it benifitial to pass above the target. However, it was found more benificial to pass the same target twice, than fly above both targets.

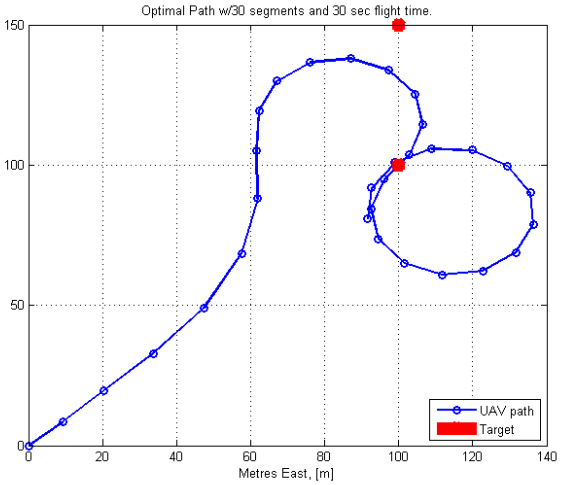
All in all, with neither of the distance minimizers did the optimal path go directly above both targets. Only with the exponential functions were a single target flewn above. It was a little surporising to see that both with the quadratic distance and norm disatance objectives, completely around the targets. The poor results can come from a



(a) Minimizing quadratic distance to 2 targets



(b) Minimizing Distance Norm to 2 targets



(c) Minimizing Exponential Distance Norm to 2 targets

Figure 4.4: Surveillance of 2 Targets using distance minimizing

combination of reasons. Since it is the total distance between UAV and targets over the whole optimization that is minimized, it is no explicitly given in the objective function that the path has to pass directly above the targets. Even though it was found that with a single target, optimal paths would go directly above targets, a combination of target distances and the turning capabilities of the UAV makes passing targets non-optimal with these objective functions.

Speed, direction and input graphs vere much the same a for a single target, and was not found necessary to present here. The UAV would typically accelerate in the beginning, then decelerate down to minimum speed at the turns, in order to perform turns as tight as possible.

4.8.2 Surveillance of 4 targets

The three distance minimizing objective functions were also implemeted with four targets. Calculated paths are seen in fig. 4.5. The most obvious is that all paths are much closer to visitng all targets when theres 4 than 2 of them. It seems that the combination of path length/optimization horizon, target patterns UAV turning raduis and objective function wheights this time made the optimal paths come much closer to the targets.

4.8.3 Evaluation of distance minimizing, parameteres and tuning

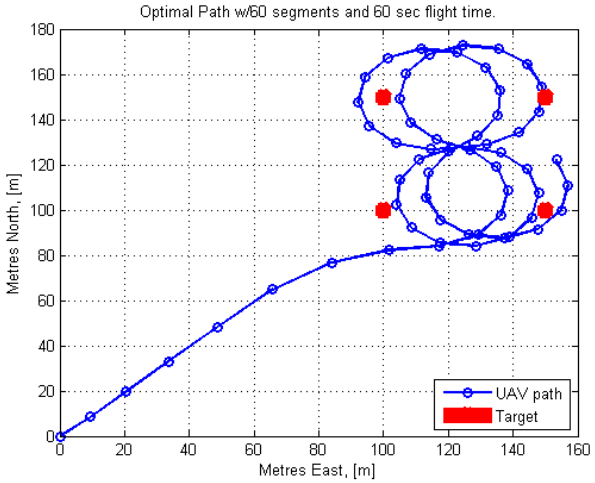
Concidering that it is wanted that a optimal path passes directly above all targets, none of these were perfect results. There was not found time to experiment further with target positions and segment lengths, but it seems reasonable that these also will have to be concidered, when searching for objectives that will make optimal paths visit all targets.

A challenge with the objective functions is that it can be found more optimal to fly close to a target twice, than directly above it once. Since boolean variables are not availible, it is hard to find a formulation that makes it unnecessary to return to a target once visited.

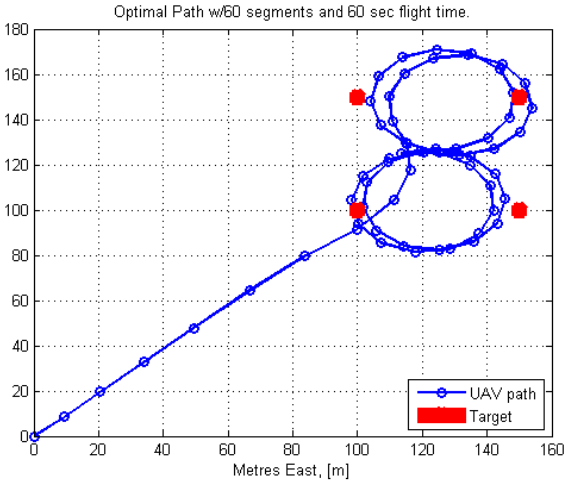
For the minimization of qudratic distance to targets, typical wheigst were $Wt = 10$, $Wr = 5$ and $Wa = 120$. With the distance norm the wheigths are typically $Wt = 10$, $Wr = 1$ and $Wa = 100$. Again; Wa might seem big compared to the others, but the sum of all acceleration inputs squared are much smaller than the sum of all distences, so the large wheight is necessary to avoid heavy ocillations on the acceleration input. With the exponential function, typical wheights were $Wt = 100$, $Wa = 1$, $Wr = 1$ and $k = 0.007$. When there were multiple targets, the targets were all given the same values for Wt and k.

4.8.4 Computational time with 4 targets

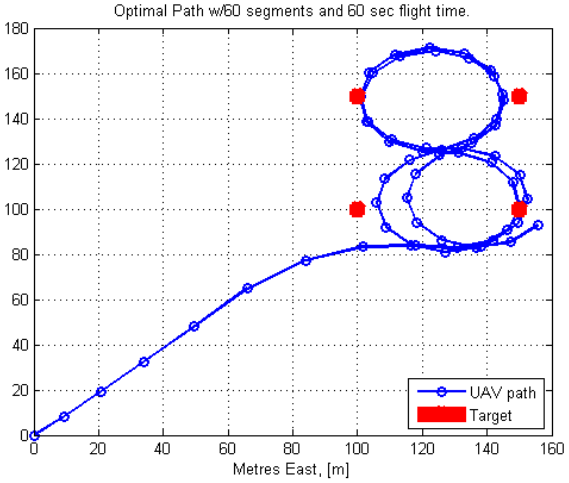
Typical computation times for the three different distance minimizing objective function with four targets are shown in table 4.4. It is noted that optimizing the simplest objective function took the longest time to calculate. The quadratic objective is said to be the simplest of the three, because the $f_N = \sqrt{f_Q}$, and $f_E = e^{-k \times f_N}$.



(a) Minimizing quadratic distance to 4 targets



(b) Minimizing Distance Norm to 4 targets



(c) Minimizing Exponential Distance Norm to 4 targets

Figure 4.5: Surveillance of 4 Targets using distance minimizing

S	T	Targets	Iterations	Time	Hessian	Solution	Minimizer
60	1	4	176	0.820	Exact	Optimal	Exponential
60	1	4	258	1.139	Exact	Optimal	Norm
60	1	4	306	1.530	Exact	Optimal	Quadratic

Table 4.4: Comparing of distance minimizers computional time

4.9 Minimizing Information Uncertainty

In these optimizations both the UAV model, eq. (3.5), and the uncertainty model, eq (3.7), are used in the discretized system model. Unce the information uncertainty at given target locations are included in the discretized system model, it is sought to minimize the toal information uncertainty. This gives rise to the tested objective function

$$f_{Uncertainty} = W_T \times f_{U_{TG}} + W_a \times f_a + W_r \times f_r \tag{4.5}$$

See implementation chapter for details.

4.9.1 Uncertainty model on single Target

Figure 4.6 shows the optimal path with a single target, where the objective is to minimize the overal uncertainty. A segment length of $T = 1$ seconds is used. The used wheigts and parameters are as follows: $k = 0.05$, $b = 5.0$,$U^{(0)} = 100$, $\gamma = 100$, $W_T = 100$, $W_a = 10$, $W_r = 1$.

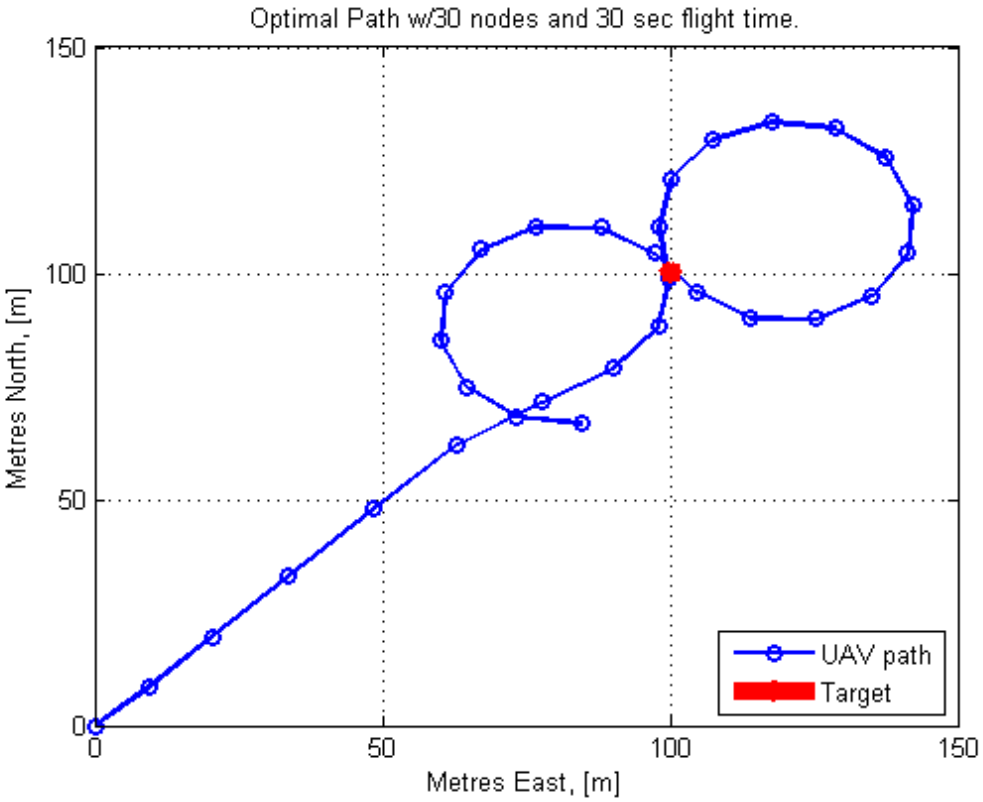
It is seen that the UAV path leads almost directly to the target and circles back to it, as opposed to the the distance minimizations in figs 4.1a and 4.3a where the path "looped around" the target at first. It is clear that the objective function with the uncertainty model gives the optimization a clear insentive to fly directly above the target, instead of circeling around it.

As for the other paths, the UAV accelerates maximum in the beginning to decrease distance between UAV and target fast. Before reaching tha target , the UAV decelerates to minimum speed, in order to perform sharper turnes. Even though a plot of inputs is not included here, the distance between the nodes (circles in the graph) in that path plot gives an indication of the speed. The longer the distance, the higher the speed.

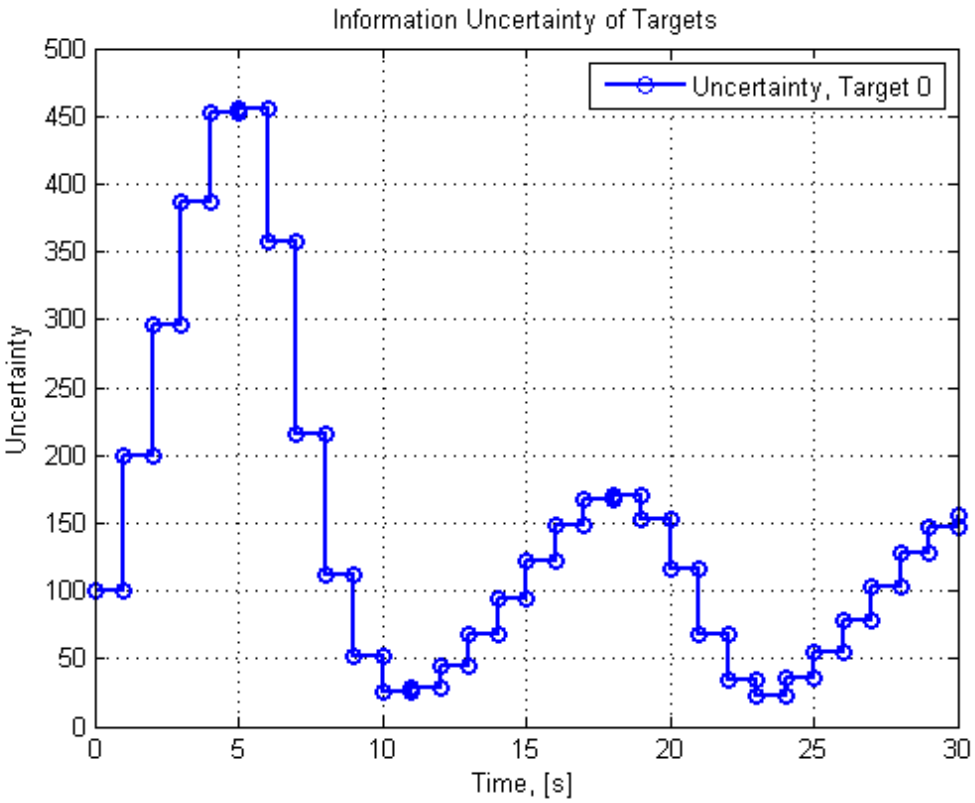
Comparing the path, fig. 4.6b, and the plot of the corresponding target uncertainty, fig. 4.6b, it is seen that the uncertainty behaves as anticipated. Because of the γ part the uncertainty rises in the beginning. This is as wanted; the uncertainty of the situation in a given target should rise whenever it is not observed. It is seen that the uncertainty rises slover as the UAV comes closer to the target, until the uncertainty flattens, and starts to decrease after 6 seconds. At that point the UAV is approxiamtly in position (80,70). After 10 seconds, the UAV passes above the target, and the uncertainty starts to rise. For repetition, the uncertainty model is

$$\dot{U} = U \times (\alpha - \beta) + \gamma$$
$$\beta = b \times e^{-k||\mathrm{UAV}_{pos} - \mathrm{Target}||_2}$$

It should be noted that in order to find satisfying paths, α had to be zero. With the chosen values for k and γ, it looks like the uncertainty is reduced when the UAV approaches the target, and starts to rise again the second the UAV passes it. This comes from comes from the large γ values used. With lower γ values, the uncertainty will continue to be reduced for a little while after the UAV has passed the target. This is because the $P \times (-\beta)$ part of the uncertainty model is negative also close to the target. However, a large γ



(a) Optimal Path



(b) Developement of the uncertainty

Figure 4.6: Surveillance of a target, minimizing uncertainty

was necesarry to get the path to go above the targets. A to large, or to small made the paths go further way from the target. Typical tuning valus were $0.01 < k < 0.05$ and $100 < \gamma < 150$. THat the uncertainty rose again after tha target was visited the first time, mande it favourable for the UAV to visit the target once again.

4.9.2 Uncertainty model on two targets

When introducing a second target, the systems is expanded with an additional uncertainty model. There is now two uncertainty models in the system, one for each target. The first target is still positioned in $(100, 100)$, while the second is in $100, 150$. Both of the uncertainty models has the same parameters as in the previous example, and also the objective function wheights are the same. The optimal path with two targets and the corresponding uncertainties are plotted in figure 4.7. Also with two targets the optimal path passes (almost) directly above the two targts. The last node of the path is only a few meters away from the target. Since the goal is to get the path to fly directly above the targets, this is a good result, compared to the distance minimizing objective functions seen in fig. 4.4. With quadratic and norm distance minimization, the optimal path circled around, the targets. The exponential distance minimizer only flew above one of the targets, where the uncertainty minimizer flies above both.

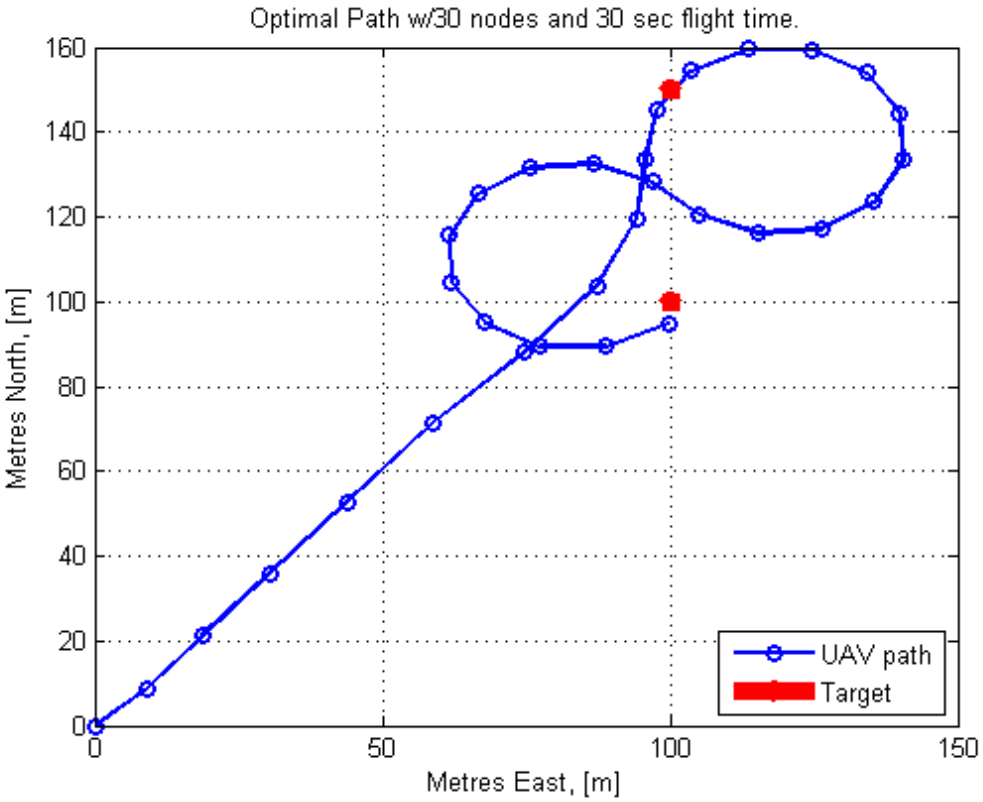
Again the results give a curve for uncertainties, fig. 4.7b that increases when the UAV is far away from the target, or heading away from them after beeing close. The uncertainty is reduces as the UAV closes in on the target. It is seen that the two uncertainty graphs oscilates as the UAV is close or distant, and they are a bit "phase shifted". This is as expected, since the UAV alternates between wich target it is closer to.

It is interesting to see that the UAV first flies close to the first target, reducing the uncertainty there, before flying to the second target, and then back again to the first. From a surveilance-point of view it seems reasonable to go back and forth between targets, to minimize the overall uncertainty.

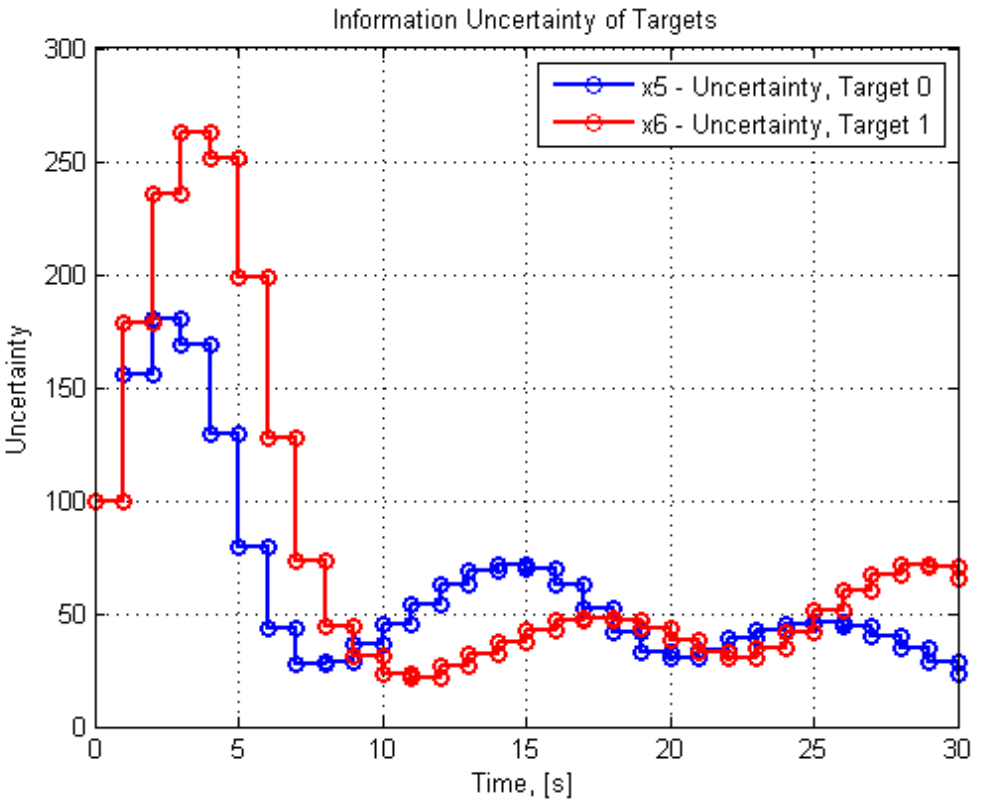
4.9.3 Uncertainty model on target grids

It is already seen that using the uncertainty model calculates optimal paths passing over two given targets. Now, results of optimizations with four and then nine targets are given in figure 4.8. When introducing grids, it is still thought that the target is an iceberg needed to be directly overflown in order to inverstigate it, for instance to assess it's size. Parameters and whights are chosen to obtain paths passing directly above as many targets as possible. With 4 targets in a 2-by-2 grid, the optimal path using uncertainty models shown in fig 4.8a takes on a "sideways figure 8" shape, where all targets are closely flewn by at least twice. The flight ends in the south-eastern node, wich is visited three times. Again it is seen that the optmimal goes almost straight towards the target closest to the UAV startpoint, before entering a circeling pattern where all targets are visited in turns.

With 9 targets in a 3-by-3 grid it was more difficoult to find parameters and wheigths making the optimal path visit all targets. The targets are so spread out that if the UAV

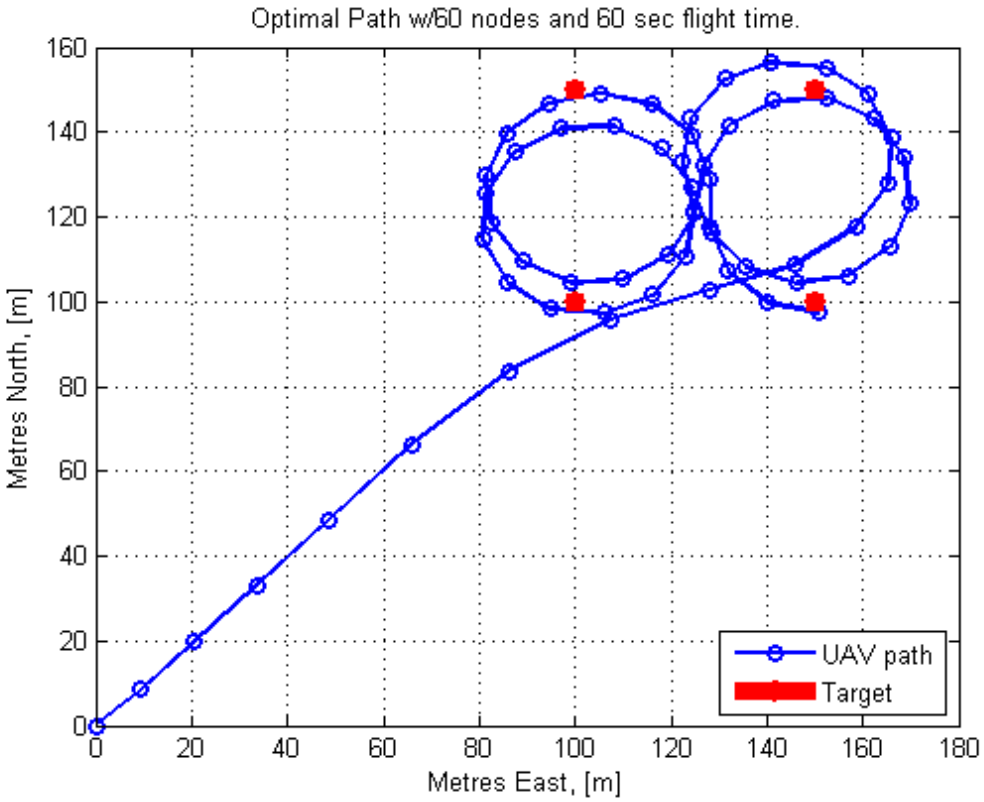


(a) Optimal Path

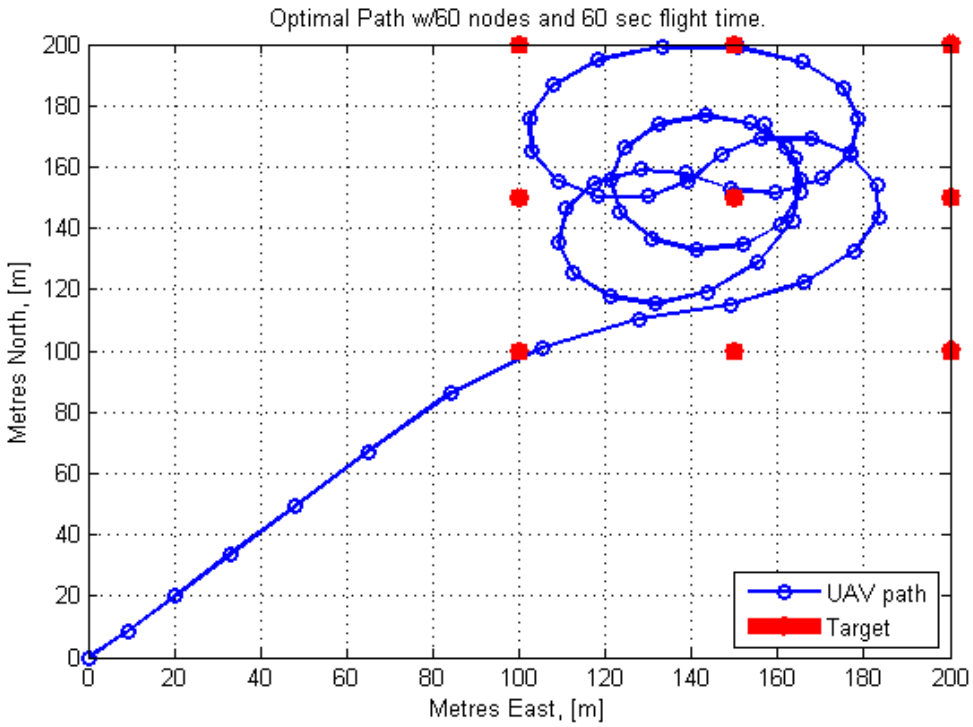


(b) Developement of the uncertainty

Figure 4.7: Surveillance of two targets, minimizing uncertainty



(a) Optimal Path in 2-by-2 grid



(b) Optimal path in 3-by-3 grid

Figure 4.8: Surveillance of grids, minimizing uncertainty

is to visit one of the corner targets, the uncertainty have time to grow large again on the opposite side. In most cases the Optimal path would make the UAV spend most of its time circeling the center target, and not come close to the corner targets, except for the down-left target. It was in most cases flown above when the path entered the grid area.

The best optimal path found, that being the one path closest to all targets, is presented in figure 4.8b. The pathe starts out with something similar to a "letter s" shape along the rows of targets without passing above them. After reachin the pint closest to the top right node, it circles around the middle target. The path only comes close to the leover left and middle target, and the lower right target is not closed in on at all.

An important reoccuring feature when using minimization of total uncertainty for path planning, is that these paths seems to go directly towards and above the targets, in the cases with 1, 2 and 4 targets. Aslo with 9 targets, the target closest to the path's starting point is flewn above when entering the grid. With distance minimizing, it was more difficoult to find parameteres that made the path fly close to the target. This is clearest in the case for 2 targets.

A clear result is that is it easier to tune the uncertainty model and the uncertainty minimizing objective function to create optimal paths visiting all targets, than the case is for optimal paths based only on distance minimizing.

4.9.4 Robustnes and tuning of uncertainty model parameters

If setting the parameter k too big, it seems to become difficoult for the IPOPT algorithm to find good search direction. A bigger k makes the "bell curve" more narrow, and it becomes more benificial to fly directly above or very close to the specified targets. However, far from the targets, both the bell curve and its gradient is zero. With a narrow bell shape, and a start point for the IPOPT algorithm that is well away from the targets, the targets might become "invisible" to IPOPT, and the optimization algorithm is unable to find a feasible, optimal path.

In the first tests, a simulation of a flight straight north was used us a start point for the IPOPT algorithm. In combination with a big value for the k parameters, the IPOPT algorithm was unable to find good search directions. The results were meaningless, very infeasible paths.

To help this, a straigth fligth in direction 1 (radian) was introduced as startpoint for the optimization. This helped the algorithm discover the targets, as the initial point is a path passing close to the targets. When selecting the size of the k parameter, there is a trade-off. Bigger k makes it more more benificial to fly close to the the targets. At the same time will a bigger k make the algorithm less robust.

It was also noted that when using a lower value for γ and k, the optimal paths for surveilance of a single or two targets was more similar to the behavoiur of the paths based on distance minimicing. The UAV tended to circle around the targets, much as in figures 4.4a and 4.4b, instead of flying directly above therm.

4.9.5 Memory in grids

When the uncertainty model vas first introduced, it was mentioned that the uncertainty model could introduce some sense of "memory" in the optimization. During test with minimizing the total uncertainty for multiple targets, it was sought to find a parameter set that would make the target uncertainty very low when visited, and then make the uncertainty rise only very slowly, or not at all. It was hoped that this wold make it unnecessary for the path to seek back towards targets, after they first was visited, possibly making the optimal path fly directly above a target at a time. The surch for such a parameter set was not succesfull, as seen from the path with 9 targets, fig. 4.8b. In order to reduce U only when very close to the targets and then not rise afterwards, k needs to be big and γ small, but such a combination was not found. It cannot be concluded that it is impossible, but it is difficoult due to the previosly dicussed robustness issue.

4.9.6 Computational times, uncertainty Model

S	T	Targets	Iterations	Time	Wheights
30	1	1	102	0.366	Wa=10, Wr=1 , Wu=100,α=0,b=5,k=0.05 γ=100
30	1	2	134	0.645	Wa=10, Wr=1 , Wu=100,α=0,b=5,k=0.03,γ=100
30	1	2	247	1.047	Wa=10, Wr=1 , Wu=100,α=0,b=5,k=0.02,γ=100
60	1	2	266	1.799	Wa=10, Wr=1 , Wu=100,α=0,b=5,k=0.02,γ=100
30	1	4	90	0.495	Wa= 1, Wr=0.1, Wu=10,α=0,b=5,k=0.02,γ=100
60	1	4	243	2.239	Wa= 1, Wr=0.1, Wu=10,α=0,b=5,k=0.02,γ=100
60	1	9	332	6.642	Wa= 1, Wr=0.1, Wu=10,α=0,b=5,k=0.03,γ=150

Table 4.5: CPU times, minimizing information uncertainty

Table 4.5 shows the CPU time and the number of iterations used to calculate the opimal paths for various number of targets and semgnets. Since the system model includes more variables needed to be optimized, it is expected that using the uncertainty modell wil give longer computational times than was the case when only the UAV model was used. For instance; finding the path with 4 targets gives 4 UAV states, 2 UAV inputs, and 4 uncertainty states pr node in the optimization. For 60 segments, that gives a total of $(4+2+4)\times 61 = 610$ optimization variables. Solving the same problem, minimizing distance instead of uncertainty gives only $(4+2)\times 61 = 366$ optimization variables. In table 4.4 it was shown that the minimizing the quadratic distance to 4 targets tok 0.820 seconds for a path with 60 segments of 1 seconds length, while using uncertainty models it took 2.239 seconds to calculate the path.

Although it is seen that also objective function wheigts, and values for k and γ has an influence (see entries 2 and 3 in the table, how changing only k from 0.03 to 0.02 increases CPU time with 0.4 seconds), the general trend is that optimizing using the uncertainty model is slower than not unsing it, especially when the 2-by-2 and 3-by-3 target grids were intruduced.

4.10 Results from MPC / On line path planing

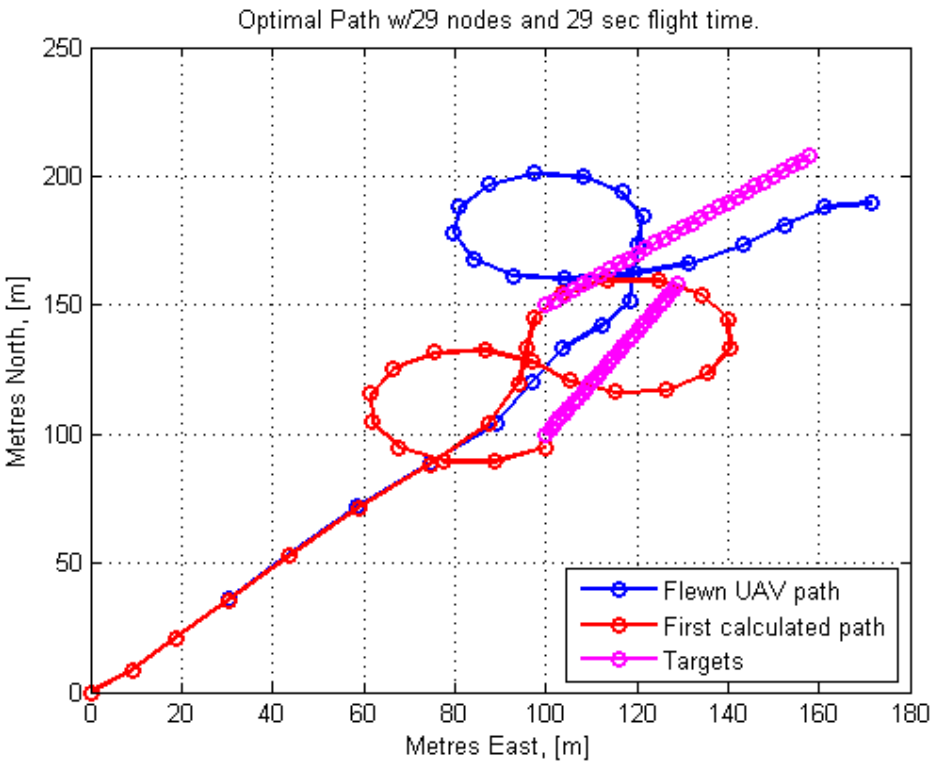
A couple of different test scenarios is set up to explore different uses of MPC/on line path planning. The results are given in the following section. For the MPC controller tests, two targets and the uncertainty models will be used. The optimization will be done for a 30 segments long horizon, with a segment length of 1 second.

4.10.1 Updated target info

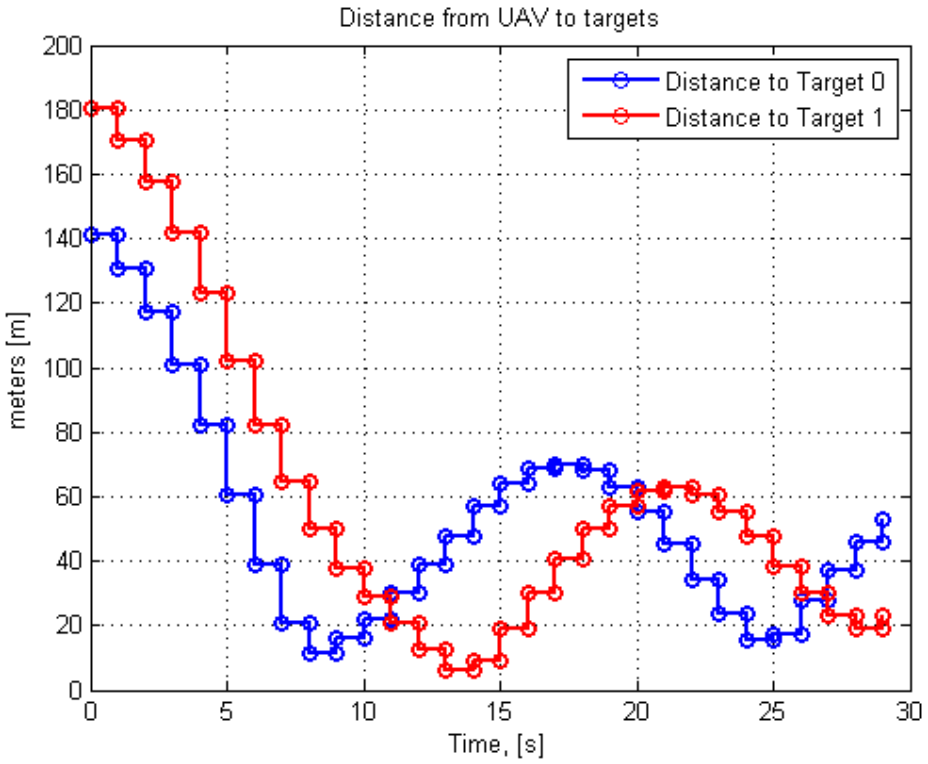
In the first test scenario it is assumed that target positions are given by a source outside the MPC controller. The optimal path is recalculated when the MPC receives updated info. In each of the solved optimization problems, it will be assumed that the target is stationary, as in the previous examples. The point is to see how the flewn path turns out to be if the target position is frequently updated.

4.10.2 Update every 1 second

In the first test, the target info is updated every second. The segment length is also a second. Therefore is the UAVs position, speed,and heading in node 1 of the first optimization is used path start point, node 0, in the next optimization.



(a) First calculated and flewn paths



(b) Distance from UAV to targets

Figure 4.9: NMPC Updating path and target every second

Target 0 starts at coordinate $(100, 100)$. At each update, it is moved 2 meters north and 1 meter left, which simulates a linear drift in north-northeastern direction of approximately 2.26 m/s. Target 1 starts in position $(100, 150)$ and is moved 2 m north and 2 m east at each iteration, simulating a northeastern drift at approx. 2.8 m/s.

The results of this first case is given in figure 4.9. In figure 4.9a, the optimal path calculated at the first NMPC iteration, when the targets are thought to be stationary at coordinates (100,100) and (100,150) are plotted in red. The actually flown path is plotted in blue. The positions of the targets are plotted in magenta.

To more easily see the distance between the UAV the targets when using the updated (flewn) path, the distances is plotted vs time in figure 4.9b.

As seen, the flewn path is almost identical to the first calculated for first 8 seconds, but after that, they differ. As before, the first calculated path seeks to fly close to the target, passing it in approx. 10 meters distance, before it passes directly above the second targets, and turns back to end the path very close to the first target. It seems that the start of the optimal paths in the first 7 iterations are similar; that all these optimizations gives path in the same general direction, passing a few meters away from Target 0. However, after first passing Target 0, the smooth "figure 8" shape seen in the first calculated path, and in most of the previous results, does not appear. The path does a left-right-left turn, while flying in parallel with both targets, as if it keeps changing mind of what is the optimal direction. When put in a position in the middle of the two targets, the path planning seems to become "indecisive", what targets the path should go to first seems to change at almost every iteration. After 12-13 seconds, when the UAV paths crosses Target 1's (the northernmost target) position, the smooth turn indicates that all consecutive iterations of the MPC finds continuing this turn optimal. After the turn, at 25 seconds, when the UAV path again is between the two targets, the "indecisive" behavior appears again.

4.10.3 Changing shape of optimal paths

This can be interpreted to mean that for the first part of the flight, while the UAV is far away from the targets, optimal paths will be quite similar, even if the start point of the path moves with a few meters. In proximity of a target, particularly when being between them, and the UAV direction on the starting node of the path does not point near any of the targets, even a small change in the UAV position of the first node will give completely different flight paths.

The result is that flown path does not come very close to any of the targets. The closest it gets to Target 0 is approximately 10 meters, after 8 seconds flight. It seems that the optimal paths keeps planning to visit Target 0 quite late in the path, and because the path is continuously recalculated, the UAV never gets there. The flown path is closer to Target 1, as the distance is down to 5 meters after 13 seconds. On closer inspection it is seen that the node 13 of the path is on the south side of Target 1's path , and the 14th node is on the north side, so with a higher plot resolution, the displayed distance would be lower, since to UAV would be closer to the target between nodes 13 and 14.

The result demonstrate a concept that is both the strength, but sometimes also the

downside of MPC control. Because the MPC plans ahead for the whole horizon, there is a possibility that the optimization will find it best to pass targets close at the end of the horizon. This could mean the flown path never will come very close to any of the defined targets, since they can be approached from any direction.

It was in the earlier results noted that the uncertainty minimizing scheme for optimal path planning tended to find optimal paths that would pass either close or directly above the nearest targets at the first opportunity, and go to the next target. This seems to be a helpful feature, when the targets are tracked. At least when the path is recalculated in a point where one target is clearly closer than the other.

4.10.4 Flight planning memory

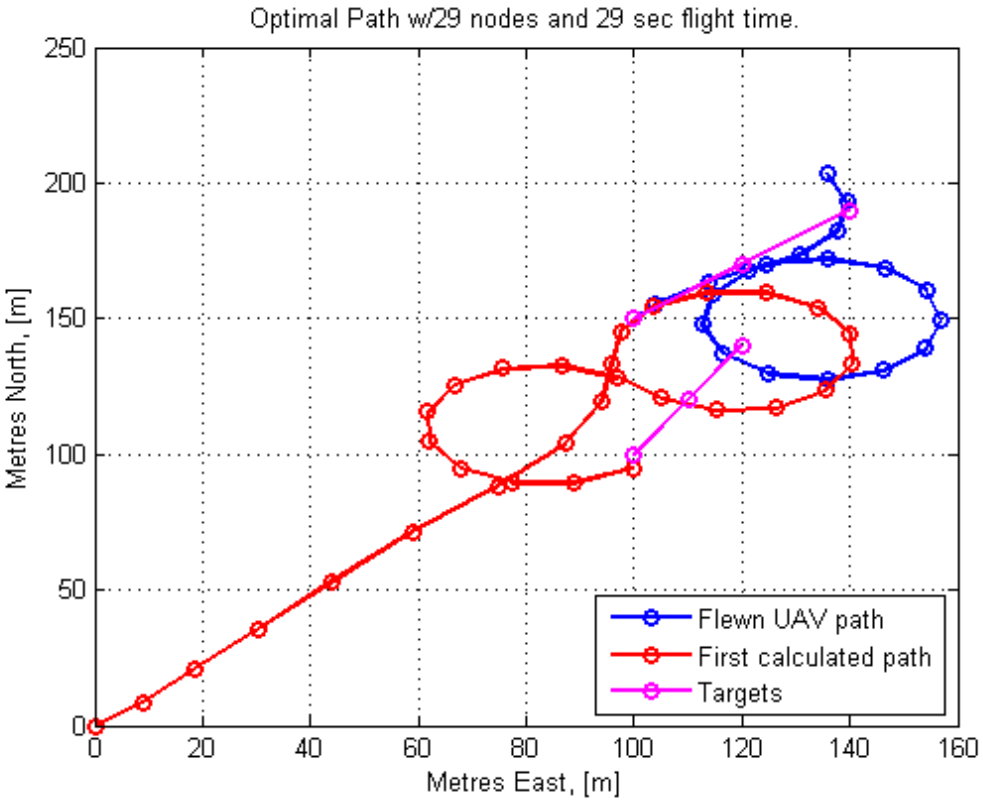
With the uncertainty model in the MPC scheme, the optimization is actually "remembering" where it has been. This is because the value of the uncertainty at the node the UAV was in when the path is recalculated is used values for node 0 in the next iteration. So when the UAV has been closer to one of the targets, the start uncertainty, uncertainty in node 0, will be significantly lower.

4.10.5 Parameter tuning

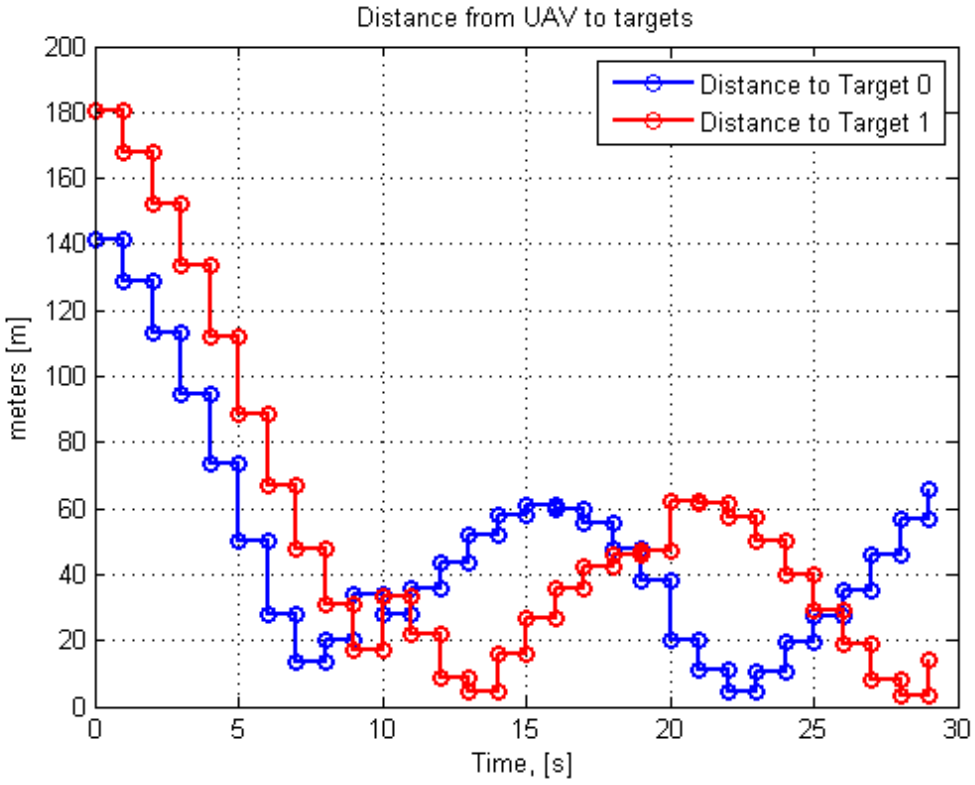
As has been noted earlier, parameters in the uncertainty model, objective function weights, segment length and optimization horizon will influence what the optimal paths will look like. Planning shorter path, where theres only time to visit the targets once, might help creating flown paths closer to the targets.

4.10.6 Update every 10 seconds

In figure 4.10, the path is updated every 10 seconds instead. The target drifts speed and directions is as above, Target 0 drifting north-northeast at 2.26 m/s, and Target 0 drifting northeast at 2.8 m/s. Now their positions are only updated every 10 seconds. Here, the UAV is given more time to close in on the targets before the path is recalculated. It is in figure 4.10a seen that the UAV comes as close as 2 meters away from the last known location of both targets. In figure 4.10b it is seen that the UAV will be closest to Target 1 after 13 seconds. However, that target information is by then 3 seconds old, and the target moves with a speed of 2.8 m/s. That means the drifting ice berg the target is meant to represent could be as much as $2 + 3 \times 2.8 = 10.4$ meters away.



(a) First calculated, and flewn paths



(b) Distanze from UAV to targets

Figure 4.10: NMPC Updating path and targets every 10th second

The UAV path is closest to Target 0's last known position after 23 seconds. Here it is 2 meters away from the assumed Target 0 position. Again, the Target position was updated at time = 20 seconds, so the ice berg Target 0 is meant to represent, might have drifted away during the 3 seconds that have passed. In worst case the target will be $2 + 3 \times 2.26 = 8.78$ meters away.

From the graphs it is seen that a lower update frequency of target position helps the path come closer to the assumed target positions. However because the ice bergs drifts continuously, the worst case minimum distance to the targets were quite similar when updating position every 1 and 10 seconds. Because the path is updated more slowly, the flown UAV path is more smooth than the flown path in the example with updates every second.

4.10.7 Wind disturbance

In section 4.5 it was seen that paths based on the discretized system might not be identical to the paths flown in reality if the calculated optimal inputs are used on an UAV. Neither does the optimization consider disturbances, such as wind. Also, system models are often inaccurate simplifications. As of this, it will probably be necessary to update the optimal path and inputs during a flight, in order to fly across the given target positions if the inputs found in the optimization is to be applied directly to the UAV actuators. The MPC controller is chosen to update/recalculate the path every 2 seconds. This is because optimization with two targets and 30 segments in most cases has taken anywhere from 0.5 to 1.5 seconds, depending on parameters and start point. As discussed in the MPC CHAPTER, it is essential that the optimization is done when it is supposed to be applied.

In this case, a NMPC controller is applied to the UAV, and the UAV is exposed to a constant wind, of 5 m/s from the north. Since the speed in the UAV model is wind speed, and it is assumed zero wind when deriving UAV ground position relative to a given point on the ground, the UAV will drift of the optimal path when exposed to wind.

This is shown in figure 4.11. The optimal path calculated in the first NMPC iteration is plotted in red. It has the now familiar "figure 8" shape, passing directly above both targets. Plotted in green is the path flown by the UAV with inputs from the first optimization, and the wind disturbance applied. For each node/second, the UAV north coordinate moves 4 meters further south than the optimization calculated. Quite frankly - the result is bad. At the point the UAV is supposed to fly above the northernmost target, it approximately 10 meters south of the southern target, and it ends up $30s * 5\frac{m}{s} = 150m$ south of where it is supposed to end.

The UAV path flown when using the MPC controller is plotted with blue in figure 4.11. It is seen that as long as Target 0 is closest to the UAV, the flown path is similar to the first calculated path. The path is just a little more south, because of the wind disturbance that made the start point of the first recalculation be 10 meters south of the anticipated position from the first calculation. As has been common when using the uncertainty model, the path goes close to Target 0 at the first opportunity, and goes to Target 1, where information uncertainty is grown higher.

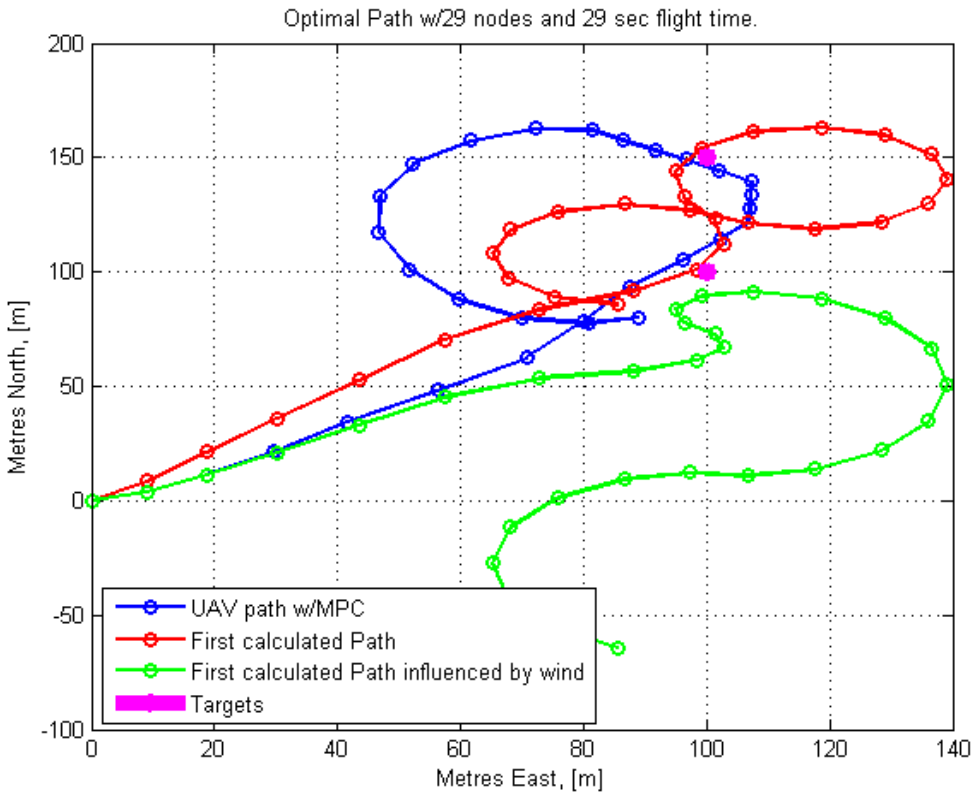


Figure 4.11: Wind disturbance: NMPC Updating path every 2nd second

Then a bit of this previously discussed "indecisive" behavior, as the path is straight for some time. This could indicate a right turn being affected by the wind, until making a left turn becomes more optimal. That the left turn becomes so wide is also because of the wind that blows the UAV southwards.

All in all, the last 22 seconds of the flight is very different from the originally calculated path. The flown path is very far from any targets,where it is unable to reduce the information uncertainty.

4.11 Hot Start and algorithm initial point

In connection with the planning of the optimal paths using the uncertainty model it was mentioned that the optimization was more robust when the start point for the algorithm is a path passing not far from the targets. If the algorithm is given a start point close to the optimal point, it should also be able to find the optimal point in fewer iteration, saving computational time. In connection with MPC, there is an idea to use the optimal solution found in an iteration as a starting point for the next. This is called hot start.

It is possible to pass on the optimal values for Lagrange multipliers and dual variables as start points for the next.

It was tried to find a solution for using hot start on the MPC controller used in the case with the wind disturbance. Since the start node of the optimal path - the UAV position,speed,and heading in node 0 - moves for each iteration, a hot start solution where the optimal values from node 2 where mowed forward to become the values in node 0 in the

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Common Public License (CPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

NOTE: You are using Ipopt by default with the MUMPS linear solver.
      Other linear solvers might be more efficient (see Ipopt documentation).


This is Ipopt version 3.8.1, running with linear solver mumps.

Number of Iterations....: 247
Total CPU secs in IPOPT (w/o function evaluations)   =      0.832
Total CPU secs in NLP function evaluations           =      0.196
EXIT: Optimal Solution Found.


This is Ipopt version 3.8.1, running with linear solver mumps.
Number of Iterations....: 149
Total CPU secs in IPOPT (w/o function evaluations)   =      0.574
Total CPU secs in NLP function evaluations           =      0.121
EXIT: Optimal Solution Found.

This is Ipopt version 3.8.1, running with linear solver mumps.

Number of Iterations....: 97
Total CPU secs in IPOPT (w/o function evaluations)   =      0.276
Total CPU secs in NLP function evaluations           =      0.068
EXIT: Optimal Solution Found.

This is Ipopt version 3.8.1, running with linear solver mumps.

Number of Iterations....: 52
Total CPU secs in IPOPT (w/o function evaluations)   =      0.133
Total CPU secs in NLP function evaluations           =      0.041
EXIT: Optimal Solution Found.

This is Ipopt version 3.8.1, running with linear solver mumps.

Number of Iterations....: 388
Total CPU secs in IPOPT (w/o function evaluations)   =      1.257
Total CPU secs in NLP function evaluations           =      0.312
EXIT: Optimal Solution Found.

This is Ipopt version 3.8.1, running with linear solver mumps.

Number of Iterations....: 3000
Total CPU secs in IPOPT (w/o function evaluations)   =     11.815
Total CPU secs in NLP function evaluations           =     16.654
EXIT: Maximum Number of Iterations Exceeded.

This is Ipopt version 3.8.1, running with linear solver mumps.|
```

Figure 4.12: Development in computational times with Hot Start MPC

It is seen that the calculation times and number of iteration drops significantly after the first iteration of the MPC controller. The first NLP is solved in about a second, the next in 0.7, then 0.35 and then 0.174 second is the lowest. However, after the fifth optimization, the start point for the next optimization became very infeasible, and the algorithm was unable to find good search direction. An optimal solution was not found, and no valid start point was given to the following iterations. More practically put; the MPC controller crashed. The large infeasibility is caused by state x_0^n being updated to be correct for the wind influence, but the uncertainty models were not. That caused the infeasible hot start point, because x_0^n is a variable in the target information uncertainty model as well.

However, this issue was not resolved due to time constraints.

The outtakes from the log file, figure 4.12, still shows that with smart initial points for the algorithm, the solution time of the NLP solved at each MPC iteration can be reduced significantly.

It also serves as an example of how important it is for a controls system to have a solution in hand if the UAV suddenly is unable to find feasible solution. Having controllers that are known to crash from time to time is simply unacceptable.

It was also seen in the results from comparing discretization with collocation and RK4, that there is a difference when using them. As stated previously, the initial point of the IPOPT algorithm is found by simulating a straight flight, given as start position, start speed and heading. All inputs in the simulation is zero. Nine targets, U(0) = 100, γ = 150, primal variable infeasibility = in the 10^3 area. Infeasibility grows with these parameters. With none, typically around 10^0. The IPOPT algorithm was able to handle these infeasible initial points. In the case where the MPC controller was unable to find an optimal point, the primal infeasibility was in the 10^{22} area. Even with increasingly infeasible hot start point, the computational time was sinking during the first few optimizations of the MPC controller.

4.12 Comparing with previous work

As mentioned in the introduction, the master thesis is a continuation of a previous fall project [15]. In the fall project, collocation was used to discretize the same UAV model as used here. Optimal paths for surveillance of a single target was found using the *fmincon* NLP solver implemented in MATLAB. The objective function for minimizing quadratic distance was used, the same as eq. 4.1, only without punishing use of rudder.

Unlike IPOPT, fmincon only needed the objective function and the constraint functions defined. It calculates gradients and Hessians needed to solve the NLP with a SQP algorithm by itself [19]. The Hessian is approximated with a quasi newton method. Neither did the fmincon take advantage of sparse matrices. A comparison of the computational times are given in table 4.6.

Solver	Type	S	T	Targets	CPU Time (s)
fmincon	SQP	30	1	1	13.980
IPOPT	IP	30	1	1	0.733
fmincon	SQP	60	1	1	250.000
IPOPT	IP	60	1	1	1.567

Table 4.6: Comparing performance of IPOPT and fmincon algorithms

The time measurements are done on the the same computer, but it should be noted that both target position and objective weights are different. In both cases the NLP solvers default values for terminating the optimization was used. It is seen that IPOPT is significantly faster; about 20 times faster with 30 segments and about 160 times faster with 60 segments. That IPOPT is faster is not surprising, for several reasons.

Most important is that c++ programs compiled for release runs a lot faster than matlab scripts. There are other other factors as well. In the theory section it was mentioned that Interior Point methods tend to solve large NLPs faster. Also important is the use of exact Hessian and sparse matrices in IPOPT which greatly reduces the number calculations when working with matrices. It is difficult to say what makes the biggest difference, since theres so many factors involved. It was noted that fmincon typically used 2000+ iterations to solve the problems where IPOPT uses less than 300, which gives an indication that much is gained by using an interior point method with exact Hessian. It was shown in table 4.3 that exact Hessians gave fewer iteration than Quasi Newton approximation in large problems. That fmincons calculation times is near 20 times doubled when gong from 30 to 60 segments, while IPOPT only doubles it's time is also an indication that sparsity format saves a lot of time. As the number of optimization variables increases, the number of zeros in the Hessian matrices grows exponentially. That gives an exponential increase in unnecessary 0×0 multiplications performed by the fmincon algorithm as well. With the sparsity format and nonzeros on the block diagonal, the number of matrix elements and calculations will roughly grow linearly.

It should also be noted that the shape of the calculated paths were much the same, as

is expected when the same UAV model and type of objective function is used. The UAV is accelerating towards the target, before it slows down and enters a ”figure 8” circling pattern at the the minimum speed.

However, it is left no doubt that a lot of computational time is saved by going from fmincon in MATLAB to IPOPT library in c++ as the base for the path planning framework.

Chapter 5

Conclusion

In this final chapter, a conclusion based on the previously presented and the discussed results is given. Some ideas for and pointer s for further work based on the results and conclusion is also given.

5.1 Conclusion

The framework with IPOPT and collocation is successfully implemented, and well suited to be used for path planning. Problem structure is exploited to initialize and solve the problem efficiently However, it lacks interfaces to other systems, like control and navigation systems, so it is not ready for practical use.

Planing paths using minimizing of information uncertainty in given targets in order to surveillance them proved very good in the cases with 1, 2 and 4 targets. The optimal path went above, or very close to the targets, which is the desired behavior. Minimizing uncertainty gave better results than the different versions of minimizing distance between the UAV and the given target(s). A trade off is that the problems with the uncertainty model has more variables, and therefore takes a bit longer to solve.

An important remark must be made about the uncertainty model. Good results were heavily dependent on parameter values and the initial point of the algorithm. Optimizing with the uncertainty model is not robust, which makes it less suitable for on line path planning /MPC, unless these issues can be resolved.

To use multiple targets to define a zone/grid to investigate, a 2-by-2 grid was the upper limit to where all grid centers (targets) were visited. In the 3-by-3 grid, the outer targets were not visited. It cannot be said for certain that this is a general limit since 50 meters between each node was the only distance tested.

On line path planning failed to produce flown UAV path very close to the targets. It seems that optimizing paths for stationary targets recursively is not a good way to track two moving targets, even if the path is updated very often.

On line path planning for surveillance of a stationary target while the UAV being subject to a constant wind disturbance produced a flewn path were the UAV wold come near to, bunt not directly on top of both targets. It was not an very satisfying results,

and it is concluded that making a controller that tracks the first calculated optimal path might perform better.

5.2 Further work

While the results found in this thesis answers several questions, there are several things to investigate further. Suggestions for further work are:

- Expand the framework to find optimal paths where a moving target is considered. The path planning minimizing information uncertainty has not been tested for such circumstances yet.
- Test the path planning with grids/multiple targets with different target distances to see if the results with multiple targets in this thesis can be generalized.
- generate feasible start points using the same collocation method as used for discretization.
- Develop interfaces for the off-line path planning to other equipment to enable "real life" use and testing.
- Expand the MPC / on line path planner with a working Hot Start function, and safety/robustness/real time features in order to use it safely.

Bibliography

[1] Multifrontal massively parallel solver (mumps 4.10.0) users' guide.

[2] Dp conference october 2009 - arctic. `http://www.dynamic-positioning.com/dp2009/session_directory_arctic.pdf`, August 2011.

[3] Unmanned aircraft system. `http://en.wikipedia.org/wiki/UAV`, December 2011.

[4] Kmb arctic dp. `http://www.marin.ntnu.no/arctic-dp/?page_id=2`, 2012.

[5] Sparce matrix. `http://en.wikipedia.org/wiki/Sparse_matrix`, May 2012.

[6] Welcome to the ipopt home page. `https://projects.coin-or.org/Ipopt`, May 2012.

[7] Lorenz T. Biegler Andreas Wächter. On the implementation of an interior-point filter line-serach algorithm for large-scale nonlinear programming. —, 2004.

[8] Yoshiaki Kawajir Andreas Wächter, Carl Laird. Introduction to ipopt: A tuttorial for downloading, installing and using ipopt. *http://web.mit.edu/ipopt_v3.8/doc/documentation.pdf*, 2010.

[9] Lorenz T. Biegler. *Nonlinear programming; Concepts, Algorithms, and Applications to Chemical Processes.* siam, 2010.

[10] Anthony M.DeLullo Lyle N. Long Albert F. Niessner Brian R. Geiger, Joseph F. Horn. Optimal path planning of uavs using direct collocation with nonlinear programming. *AIAA Paper No.2006-6199, AIAA GNC Conferance, Aug., 2006*, 2006.

[11] Rivest Cormen, Leiserson and Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 2009.

[12] P.J. Enright and B.A. Conway. Direct trajectory optimization using collocation and nonlinear programming. *AIAA J. Guidance, Control and Dynamics*, 10, 1987.

[13] T. I. Fossen. *Mathematical Models for Control of Aircraft and Satellites.* Dept. of Engineering Cybernetics, 2011.

[14] Thor I. Fossen. *Marine Craft Hydrodynamics and Motion Control.* John Wiley and Sons, Ltd, 2011.

[15] Lars A Grimsland. Uav path planning for ice intelligence purposes using nonlinear programming, 2011.

[16] C.R. Hargraves and S.W. Paris. Optimal finite-thrust trajectories using collocation and nonlinear programming. *AIAA J. Guidance, Control and Dynamics*, 10, 1987.

[17] S. Løset J. Haugen, L. Imsland and R. Skjetne. Ice observer system for ice management operations. *Proc. 21st Int. Offshore (Ocean) and Polar Eng. Conf., Maui, Hawaii, USA, June 19-24, 2011.*, 2011.

[18] J. M. Maciejowski. *Predictive Control with Constraints.* Prentice Hall, 2002.

[19] MathWorks. Product documentation, fmincon. `http://www.mathworks.se/help/toolbox/optim/ug/fmincon.html`, December 2011.

[20] M. Mejlader-Larsen and D. A. Onischchenko. Ice management operations. `http://tksneftegaz.ru/fileadmin/files/ru/documents/barenc_2020/4_phase/RN06_Ice_Management_mej_9_12_10.pdf`, August 2011.

[21] D. Philips N. A. Jenssen, S. Muddesitti and K. Backstrom. Dp in ice conditions. `http://www.dynamic-positioning.com/dp2009/arctic_jenssen.pdf`, August 2011.

[22] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer, second edition, 2006.

[23] Jan Tommy Gravdahl Olav Egeland. *Modeling and Simultaion for Automatic Control.* Marine Cybernetics, 2002.

[24] Rohlen. Relationship between ice-management and station keeping in ice. `http://www.dynamic-positioning.com/dp2009/arctic_rohlen_pp.pdf`, 2011.

[25] Artsiom Stalmakou. Uav/uas path planning for ice management information gathering, 2011.

[26] Tore Stensvold. Jobber for å kontrollere isen. *Teknisk Ukeblad*, 25, 2011.

[27] United States Geology Survey. 90 billion barrels of oil and 1,670 trillion cubic feet of natural gas assessed in the arctic. `http://www.usgs.gov/newsroom/article.asp?ID=1980`, August 2011.