NTNU

Norwegian University of
Science and Technology

Combining Methods of Mathematical Optimization and Artificial Intelligence for Autonomous UAV Mission Planning and Execution

Henning Andre Åsgård

Master of Science in Cybernetics and Robotics
Submission date: July 2017
Supervisor: Tor Arne Johansen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

In this thesis, different solutions for autonomous mission planning and execution aimed at Unmanned Aerial Vehicles (UAV) is investigated using methods of Artificial Intelligence (AI) and Mathematical Optimization. Different configurations of these solutions is then implemented as an Autonomous Mission Planning and Execution (AMPE) system in order to test the online performance of the complete system, and the different individual models that composes the system. Inspired by [26] the system is divided into modules handling the plan in different levels of refinement in a hierarchical manner. Where applicable, modules will be tested and compared when implemented by AI and Mathematical Optimization and the advantages and disadvantages will be discussed. The system should be able to plan with a certain amount of resources in mind, continuously update the plan using state feedback, and perform the plan by using the plans current actuator inputs. In order to keep clear goals for the the AMPE system, it is designed with a specific case in mind. Both the implemented capabilities and the discussion of results will focus on off-shore data gathering from sensors where the distances causes the required bandwidth impossible to achieve for direct communication. Several UAVs will cooperate in order to collect the data by creating a delay-tolerant communication network, and will seek to do so in some optimal manner. The different levels of the planning problem will be solved using Nonlinear Model Predictive Control (NMPC), Mixed Integer Linear Programming (MILP) and domain models for EUROPA. The tools ACADO, AMPL, EUROPA and T-REX will be used to implement the AMPE system with parallel, synchronous deliberation of the planning levels. Simulations will be performed which shows that using the MILP optimization with data flow constraints are note applicable for a real world system as it uses hours to solve a plan. However, by finding plans in a static mission, the EUROPA solvers will show that it is much more efficient and ways of combining EUROPA and MILP in the future will be discussed. Simulations is done to show re-planning capabilities when the mission is affected by poor flight dynamics, dynamic resource consumption and dynamically positioned nodes. In the end we will see how this system can be quite easily expanded so that the case of this thesis is only a specialization of a much broader domain of problems solvable by a more general system. A system where a mission problem would be defined as a top-level nddl domain model, and then solved automatically throughout the AMPE pipeline. The thesis will conclude with how a system based on purely Mathematical Optimization is ill-fitted for real-time demands, and how AI and Mathematical Optimization can be combined for autonomous mission planning and execution.

Abstrakt

I denne oppgaven vil forskjellige løsninger for autonom oppdragsplanlegging og utførelse siktet mot ubemannede fly bli utforsket ved å bruke matematisk optimalisering og kunstig intelligens. Forskjellige konfigurasjoner for disse løsningene er så implementert til et autonomt system for å teste den online ytelsen til det komplette systemet, og til de individuelle modulene som utgjør systemet. Inspirert av [26] vil systemet bli delt inn i moduler som behandler planen i forskjellige grovhetsgrader på en hierarkisk måte. Der det er mulig vil moduler bli testet og sammenlignet med hverandre når de er implementert med kunstig intelligens og matematisk optimalisering, og fordeler og ulemper vil bli diskutert. Systemets skal kunne planlegge med en bestemt mengde tilgjengelige ressurser, kontinuerlig oppdatere planen ved å bruke status tilbakemeldinger, og gjennomføre planen ved å bruke de nåværende kontrolverdiene i planen for flykontroll. For å holde klare mål for systemet vil det bli utviklet for en spesifikk case. Både den implementerte funksjonaliteten, og diskusjonen av resultater vil fokusere på off-shore datasamling fra sensorer hvor avstandene er for store for direkte kommunikasjon. Flere fly vil samarbeide for å samle data ved å skape et forsinkelsestolerant relénettverk, og vil prøve å gjøre så på en optimal måte. De forskjellige delene av planleggingen vil bli gjennomført ved å bruke Nonlinear Model Predictive Control (NMPC), Mixed Integer Linear Programming (MILP) og domenemodeller for EUROPA. Verktøyene ACADO, AMPL, EUROPA og T-REX vil bli brukt for å implementere systemet med parallell synkron overveielse av planleggingsnivåene. Simuleringer vil bli gjennomført som viser at MILP optimaliseringen med dataflyt begrensninger kan ikke brukes for systemer i den virkelige verden da den er treg og kan bruke flere timer på å finne en løsning. Ved å finne planer for et statisk oppdrag vil EUROPA løseren vise at den er mye kjappere enn MILP modulen, og måter for å kombinere EUROPA og MILP vil bli diskutert. Simuleringer er så gjort for å vise re-planleggingsevnene til systemet når oppdraget er påvirket av dårlig flydynamikk, dynamisk ressursbruk, og dynamisk plasserte noder. Til slutt vil vi se hvordan dette systemet kan utvides til å takle et mye bredere spekter av oppdrag. Oppgaven vil konkludere med at et system basert rent på matematisk optimalisering vil passe dårlig for ekte-tid systemer, og hvordan kunstig intelligens og matematisk optimalisering kan kombineres for autonom oppdragsplanlegging og løsning.

Contents

Part I

Introduction

1.1 Introduction

The application of Unmanned Aerial Vehicles (UAVs) have over the past decades become so significant that in 2011, International Civil Aviation Organization started regulations of civil UAV usage[10], and global guidelines have been created as templates for national laws. More and more large area both civil and military operations are being executed by remotely controlled or autonomous UAVs. Therefore, the field of UAV control and autonoumisation is an important subject, and different approaches for autonomous systems are receiving a lot of research. As the vehicles become autonomous, some important aspects of design appears. One would want the system to be reactive to changes in the environment and able to complete the mission with minimal human interaction. It is also important that the system is efficient in the sense that it accomplishes the mission in some optimal way. A mission must then be planned and executed in accordance with these requirements, and the approaches of this is often divided into control, mathematical optimization, and Artificial Intelligence (AI). By defining optimality for the mission, mathematical optimization provides a way to plan and find control inputs by this definition. The solution, if one is found, is guaranteed to be optimal within some predefined limit[23], and this optimality can be proven. It is however often a computationally expensive task, and intricate planning problems can take even high-end industrial systems a long time to solve. The AI systems on the other hand, does not give any proof of optimality. It is a well suited approach for on-line mission planning and replanning as it is designed to be responsive by maintaining the ability to decide changes of action on the fly. The approach also makes the system able to deliberate on new situations and handle unforeseen events.

In this thesis mathematical optimization for path planning and vehicle control is implemented in an ”AI environment”, meaning a program flow built around driving an abstraction hierarchy of deliberate AI modules. This system will be heavily inspired by the work of *Fredéric Py et. al.* and utilizes the Teleo-reactor executive (T-REX) system presented in [22] and [26]. T-REX implements such a hierarchy where different AI modules deliberate on different abstraction levels of a plan and on the execution of this plan. Here, some of the AI modules will be replaced by optimization program modules and see if this combination allows for optimal control, and on-line path optimization - an operation with a complexity and with a required processing time that makes it previously impractical for on-line use. I will also seek for optimal control in this environment, using a model predictive control.

1.2 Structure of this thesis

First, previous work on T-REX and optimal path planning for UAV control, as well as preliminary work done for this thesis will be presented. Then the case which will be used to test the system is explained. A high level overview is then shown of the complete system that will be developed will then give the introduction to what is required and which methods will be explored. A nonlinear and linear mathematical model of the UAVs are derived in chapter 2. The basics of deliberative control be EUROPA and T-REX will then give the basis needed to understand the approach and terminology used. Then all the T-REX and all the individual modules that constitutes the mission planner and executioner are detailed along with the methods and concepts used by these modules will be detailed. Finally external forces and noise will be touched upon before the simulation results, discussion and conclusion are presented.

1.3 Previous work

In [28] some preliminary work were done for designing the system. A non-linear model were derived, a model for a crude path planning for several UAVS using [7], and a model predictive control were designed for control and path refining. The goal of [28] was to test these approaches and several simplifications were made. This thesis will use and build on this work in order to design a complete system. The models designed in [28] applicable for the complete autonomous system will be explained in this report for completeness, and simplifications will be rectified.
This thesis will expand on the MILP path planning program with anti-collision and connectivity constraints, design a new more reliable MPC and reference trajectory, redesign some of the models using artificial intelligence for performance comparison, include considerations of wind and sensor noise, design T-REX agents for parallel synchronous execution of the modules, and simulate communications relay missions using the autonomous system.

The four main components of this thesis are the Mixed Integer Optimization Program, the Model Predictive Control, the EUROPA domain model, and the T-REX implementation of a parallel synchronous system. In *Motion- and communication-planning of unmanned aerial vehicles in delay tolerant network using mixed-integer linear programming* [17], Esten I. Grøtli and Tor A. Johansen presents a complete system for optimizing the path of several UAVs in order to create a communication network with the UAVs capable of ferrying data, by storing it in a buffer. This is much of the basis in my case and the program of this article will be the main component in the optimized path planning. [6], [8] and [18] provide formu-

lations, detailing some parts of the model even further. *T-rex: A model-based architecture for auv control, Planning and Plan Execution for Real-World Systems* explains the problems of popular control techniques where the UAV is driven by a fixed sequence of commands, and seek to fix this by the use of a teleo-reactive executive (T-REX) artificial intelligence system. The article then focuses on what the T-REX are and how it works. The more updated work, of some of the same authors, *"Towards deliberative control in marine robotics"*[26] details more of this system, and also the EUROPA artificial intelligence system. The EUROPA system is presented as a core component in T-REX. These will be the main sources used in this thesis, and the planning and architecture takes great inspiration from these.

1.4 Case

The problem to be explored is defined as a dynamic planning and a control problem. By the use of multiple UAVs, a data relay network should be formed in order to retrieve sensor data from several stationary or moving marine vessels. The vessels will throughout this report be defined as "nodes". The area of operation will be too large for direct communication with satellite links such as Iridium. The mission must be planned and executed in some optimal manner, which are adaptive to changes in node positions. Resources should be monitored and the plan should change if these are inadequate for further operations. The resources monitored will be fuel, and should this be inadequate, the UAVs should return and reach the base station before running out. The mission to be simulated is one data retrieval loop, ending with all UAVs at the base station.

To summarize, a system should be built which takes as input the position stream of moving ships or other nodes, and perform on-line planning and execution of the mission to retrieve sensor data from all nodes, while being able to "ferry" data, and managing resources.

1.5 Assumptions and scope

This thesis will take into consideration external forces but noisy signals is removed from the scope. There will throughout this work be assumed an external state-estimator and therefore fairly accurate full-state feedback. The effect of wind on the aircraft will however be taken into the system and measures to counter-act this will be implemented. The mission is assumed to be performed off-shore, where a simple bounding box is sufficient to restrict the mission area and avoid crash landings.

1.6 The AMPE system high level overview

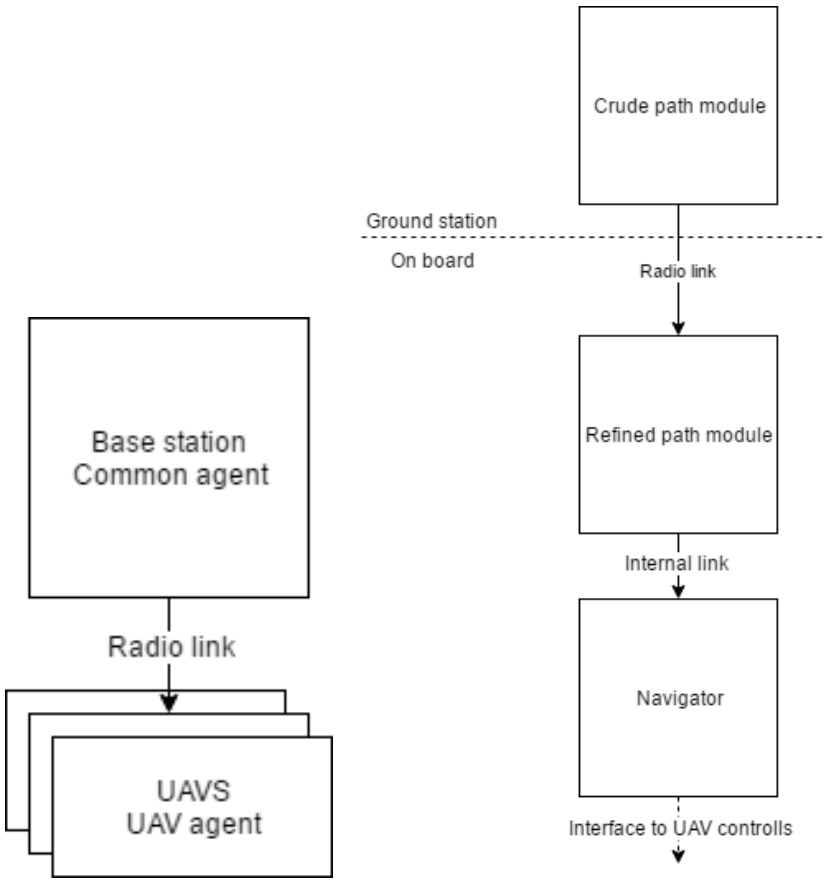
The goal of this thesis is to investigate and develop means of autonomously planning and execution of a data collecting mission involving several cooperating UAVs. This immediately poses several challenges. First, the mission should be planned and executed in some optimal manner. This optimality will be defined here as the minimization of the time duration of the mission, resource consumption and UAV actuators wear and tear. Secondly, the UAV decisions should be coupled by cooperation. The mission is intended to be performed over a large area requiring a long range, low bandwidth radio link, and the system must therefore allow for common decision between the UAVs with minimal strain on this link. Lastly, the planning must be performed on-line, as the mission is dynamic. The computational cost of the implementations then becomes important, as the deliberations should happen within some time frame. In addition to being able to retrieve the sensor data as presented in the case description, this will constitute the main requirements of the autonomous system.

The system will be broken down into agents, which are the different separate processes, and into modules, which are the different task or subproblems of the objective. Methods of implementing these modules will be explored by the use of Artificial Intelligence and Mathematical Optimization, and be compared against each other. The system in this thesis will seek to meet the requirements posed above as best as possible and the methods will be assessed by how they meet these requirements.

1.7 Agents and modules

The planning problem will require both common and individual decisions by the UAV planners. The UAVs must cooperate to solve the mission in a sensible way, and must take into consideration the individual state of the UAVs in the planning and execution. Therefore the system is split into two processes, or agents. The common agent is located at the base station and, as the name suggest, is common to all UAVs. The second agent is individual to all UAVs and is located aboard each vehicle. This is illustrated in figure 1.1a. The setup makes communication between each UAV negligible and makes the joint decision without the delay of communication between each agent. When this part has executed, each vehicle receives their relevant part of the plan. This can be done in form of sending waypoints which is a lightweight transmission, and with enough spacing of the waypoints, tolerant to the delays of an Iridium link.

The planning problem of this case is solved by splitting the system into into subproblems, or



(a) The two agents of AMPE

(b) The main modules of the AMPE pipeline

modules which are able to independently perform some part of the problem using the results of overlying modules. These modules are shown in figure 1.1b. The system developed in this thesis will be called the Autonomous Mission Planner and Executive (AMPE) and the signal flow through the modules illustrated in figure 1.1b will be the AMPE pipeline.

1.8 External libraries and equipment

The system is written as a C++ application, and several external libraries are used. These will be presented here. The optimization program for path planning, will be modelled using AMPL, and AMPL API is used as the interface to C++. AMPL is short for A Mathematical Programming Language which is a algebraic modelling language, that facilitate for the declarations of optimization programs and interface to a range of different solvers. The MILP module for path planning in section 5.2.3 is implemented as a model in AMPL as it supports mixed integer programming. The solution is found using the CPLEX solver developed by IBM. AMPL is a declarative language that are very similar to the syntax of mathematical

notation, and the model is implemented in the same fashion as the mathematical model. For the MPC ACADO Toolbox is used. This is a C++ library providing modelling interfaces, integrators and optimal control solvers. EUROPA is used for the AI module, which are developed by NASA and T-REX is used to build the overall system architecture. These will be detailed more later in this thesis.

Part II

Methods of implementation

CHAPTER 2

Deriving a linear and non-linear model for the X-8 UAV

2.1 Motivation

In order to perform simulations, as well as constraining the model predictive control later in this thesis, the equations of motion of the UAVs must be found. For the different modules and the simulation, the models used will vary. Some work will be mentioned using the Simulink simulator developed by *Kristoffer Gryte* in [16], which will use the nonlinear model presented in this chapter. This model is fairly accurate but because of performance issues, the model is linearized before applied to the MPC in chapter 5.3. The model used in the MILP formulation in section 5.2 is fairly simple and will not be presented here.

The UAV mathematical model will be designed with the X-8 Skywalker flying wing in mind. The modelling of the actuators will be especially important in this matter as the UAV lack a tail with elevators and rudder. Using the approach and equations described in [7] the reference frames used will be presented first, then the kinematics and dynamics of the UAV before a nonlinear model is completed. The linearization scheme of this model with the resulting linear equations of motion concludes this chapter.

Figure 2.1: The X-8 flying wing UAV

2.2 Reference frames

Throughout this report, the states of the UAV will be denoted with respect to one of the following reference frames. The state variables defined in these frames are listed in table 2.3.

2.2.1 NED frame

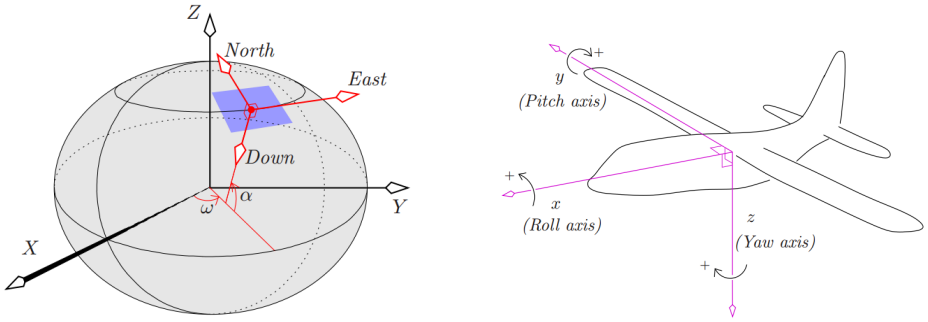
North-East-Down frame is illustrated in figure 2.2a. It is fixed in the Earth-Centered-Earth-Fixed frame which has its centre in the middle of the earth and rotates with it. The NED frame has its origin somewhere on the planet surface and moves with this location. As the name suggest, it has a axis pointing north, east and downwards. This frame will be assumed as an inertial frame, meaning we consider it to have no accelerations. The frame will be denoted by the superfix $\mathcal{F}^i$.

2.2.2 Body frame

The body frame, illustrated in figure 2.2b is defined in what is called the stability frame [7]. The stability frame has the origin fixed to the UAV but the orientation of the axes remains as the NED frame. The body frame is rotated in this frame to define the axis $\mathbf{i}^b$ out the nose of the UAV, $\mathbf{j}^b$ out the side of the UAV and $\mathbf{k}^b$ out the belly.

2.2.3 Vehicle frames

When rotating the body frame to the NED frame, some intermediate frames $\mathcal{F}^v$ and $\mathcal{F}^{v1}$ becomes important. The vehicle frame $\mathcal{F}^v$ has its origin at the UAV center of mass, but oriented as the NED frame, and then rotated by the yaw of the UAV about the axis k^i.The vehicle frame 1 $\mathcal{F}^{v1}$ gives frame $\mathcal{F}^v$ rotated by the pitch about the axis j^v.



(a) (b)

Figure 2.2: The two reference frames used throughout the report. (a) The NED frame located in the Earth-Centered-Earth-Fixed frame [20]. (b) The body frame [20].

Name	Description
u	Body frame velocity along $\mathbf{i}^b$
v	Body frame velocity along $\mathbf{j}^b$
w	Body frame velocity along $\mathbf{k}^b$
ϕ	Roll angle of body frame with respect to $\mathcal{F}^{v1}$ about $\mathbf{i}^{v1}$
θ	Pitch angle of body frame with respect to $\mathcal{F}^{v}$ about $\mathbf{i}^{v}$
ψ	Yaw angle of body frame with respect to the body frame about $\mathbf{i}^b$
p	Roll rate about $\mathbf{i}^b$
q	Pitch rate about $\mathbf{j}^b$
r	Yaw rate about $\mathbf{k}^b$
X	North position in NED frame
Y	East position in NED frame
Z	Down position in NED frame

2.3 State vector

When modeling the UAVs we are interested in the linear speed in the body frame as well as the attitude of the body frame with respect to the NED frame. The UAVs have to navigate in the NED frame, so the north, east and down positions will be the last states. The final state vector is $\mathbf{X}$, and will be stated here for convenience and as a reference for further reading. The states are illustrated in figure 2.3. The next sections will derive the equations of motion for the UAVs and gives the relationships of these states.

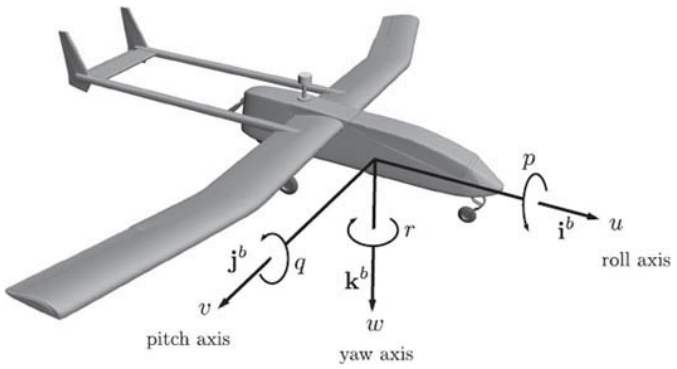


Figure 2.3: Illustration of states with the associated axes in the body frame[7]

2.4 Kinematics

For the model, the translational velocities are defined in the body frame as V_b. But in order to follow a trajectory set in the NED frame, we measure the translational positions $(N, E, D)^T$ in this frame. The position is then achieved from the body frame translational velocities by differentiation and rotation according to the vehicle attitude with respect to the NED frame. The differentiated $(N, E, D)^T$ equals the rotated linear body velocities, and the rotation are achieved by

1. Rotating V_b by $R_{D\psi}$ - ψ about the D axis
2. Rotating V_b by $R_{E\theta}$ - θ about the new E axis
3. Rotating V_b by $R_{N\phi}$ - ϕ about the new N axis

These three angles are called the Euler angles, and we get the relation.

$$\frac{d}{dt}\begin{bmatrix} N \\ E \\ D \end{bmatrix} = R_b^i \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.1}$$

We can see from the listing of the Euler rotation above, that the steps in this rotation iterates the system through the frames described in 2.2. After rotating $\mathcal{F}^i$ by $R_{D\psi}$ we get the $\mathcal{F}^v$ frame, which we rotate by $R_{E\theta}$ to get the frame $\mathcal{F}^{v1}$, which is finally rotated by $R_{N\phi}$ to get $\mathcal{F}^b$. This means that the differentiated angles are represented in their respective separate frame. In order to get the angular velocities in $\mathcal{F}^{v1}$ we get

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R^b_{v1(\phi)} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + R^b_{v1(\phi)} R^{v1}_{v(\theta)} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.2}$$

$$\begin{aligned} R^b_{v1(\phi)} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \\ R^{v1}_{v(\theta)} &= \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & -\cos\theta \end{bmatrix} \end{aligned} \tag{2.3}$$

2.5 Dynamics

Newtons second law for translational motion states that [7]

$$m\frac{d\mathbf{V}_g}{dt_i} = \sum f_n \tag{2.4}$$

m being the mass of the vehicle, $\frac{d\mathbf{V}_g}{dt_i}$ being the derivative of the speed over ground in the inertial frame, and the right hand side being the sum of all forces. Equivalently, for rotational motion, Newtons second law gives that the change of angular momentum equals the sum of all external moments **m**

$$\frac{d\mathbf{h}^b}{dt_i} = \sum \mathbf{m}_n \tag{2.5}$$

with angular momentum defined as the product of angular velocity $w^b_{b/i}$ of the bodyframe relative to the inertia frame, and the inertia matrix J

$$\mathbf{h}^b = \mathbf{J}\mathbf{w}^b_{b/i} \tag{2.6}$$

By using these equations for angular and linear accelerations of the UAVs with the forces and moments above, we can get the model of the system dynamics and kinematics (see [7] for full derivation)

The external forces acting on the UAV is the gravitational force F_g and the aerodynamic forces. There will also be the added force of propulsion detailed in the next section. The

aerodynamic forces are caused by the UAV movement through air, as a pressure distribution is built up about the UAV. The force is split into lift F_l, drag F_d and a moment m, and as the dynamic distribution is given by $\frac{1}{2}\rho V_a^2$, the foces are commonly expressed by [7]

$$\begin{aligned} F_l &= \frac{1}{2}\rho V_a^2 S C_l \\ F_d &= \frac{1}{2}\rho V_a^2 S C_d \\ M &= \frac{1}{2}\rho V_a^2 S c C_m \end{aligned} \tag{2.7}$$

Where ρ is the air density, V_a is the speed through air, S is the planform area of the UAV wing, c is the mean chord of the wing, and C_l, C_d, C_m is aerodynamic coefficients.

The drag force acts counter to the velocities of the UAV and can be split into parasitic drag and induced drag. The parasitic drag caused by the shear stress of the air moving along the UAV exterior, and pressure resistance. It is proportional to the square of the airspeed but simplifying according [7], it is assumed roughly constant around the operational airspeed as as C_{d_p}. Induced drag is also called the by-lift-induced-drag and is proportional to the square of the lift force. The drag coefficient can then be expressed as

$$C_D(\alpha) = C_{D_p} + \frac{(C_{L_0} + C_{L_\alpha}\alpha)^2}{\pi e AR} \tag{2.8}$$

The gravitational force is linear as we assume symmetric weight distribution of the UAV, and acts along the NED down-axis.

The external forces acting on an UAV airfoil is illustrated in figure 2.4.

2.5.1 Actuators

As the X-8 is a flying wing UAV, it only has two actuators

1. Throttle - The propeller gives forward momentum in the body frame
2. Ailerons - The control surfaces on the wings is used to control the attitude.

By this configuration we cannot control the yaw ψ of the UAV directly, we have to use a bank-to-turn pilot. This means that we must change the roll, and then pitch in order to turn the plane. The pitch is controlled by the actuator action δ_{as} which is the sum of the aileron deflections, and the roll is controlled by the actuator action δ_{ad} which is the difference between the aileron deflections on the two wings.

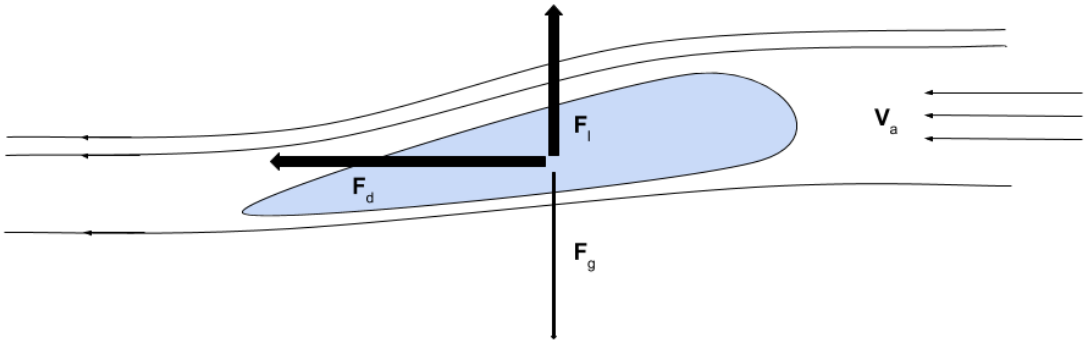


Figure 2.4: The illustration of a wing shows the forces acting on a gliding UAV body. The thin lines represent the airflow, the thick arrows are the aerodynamic forces of drag and lift, and the thin arrow is the gravity.

The thrust force from the propeller is defined by the propeller speed V_p and its aerodynamic lift. In figure 2.5 it is shown that the approach vector of the air towards the blade is dependent on the angle of attack and the airspeed of the UAV. Therefore the aerodynamic thrust coefficient becomes

$$C_{l_t}(\alpha_p) = C_l(\alpha_p)\frac{1}{2}\rho U_0^2 S \tag{2.9}$$

where α_p is the propeller angle of attack.

This gives us the thrust

$$f_t = \frac{1}{2}\rho V_p(\delta_t)^2 S C_{l_t}(\alpha_p) \tag{2.10}$$

where the propeller speed V_p is given by the thrust input δ_t.

The pitching moments of pitch and roll is given as the aerodynamic contributions of the control surfaces, the ailerons, as they are deflected.

$$\begin{aligned} M_\theta &= \frac{1}{2}\rho V_a^2 S_a c_a C_m(\alpha_a(\delta_a s)) \\ M_\phi &= \frac{1}{2}\rho V_a^2 S_a c_a C_m(\alpha_a d(\delta_a d)) \end{aligned} \tag{2.11}$$

where S_a and c_a are the platform area and mean cord of the ailerons, and α_a is the ailerons angle, and $\alpha_a d$ the difference in the angle of the two ailerons.

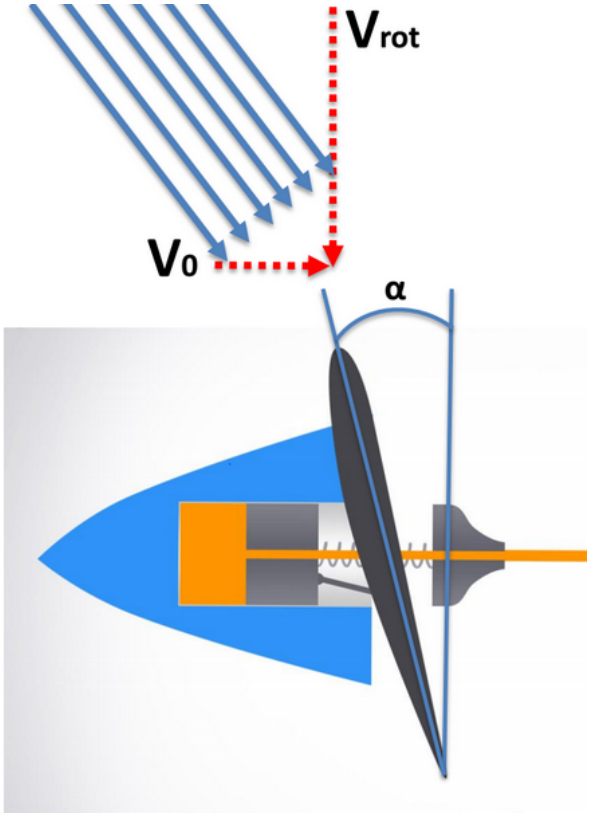


Figure 2.5: Illustration of propeller thrust. The approach vector of the air towards the blade is dependent on the angle of attack and the airspeed of the UAV. NTNU.

2.6 Nonlinear equations of motion

By combining the equations presented in this chapter, the nonlinear model of the UAV is completed. This is done in [7] and [27], and the nonlinear equations of motion is presented here. For the step by step recipe for this, the reader is referred to [7]. Note that the control signals δ_s and δ_d are modeled as δ_e and δ_a.

$$
\begin{aligned}
\dot{u} &= rv - qw - g\sin\theta + \frac{f_x}{m} \\
\dot{v} &= pw - ru + g\cos\theta\sin\phi + \frac{f_y}{m} \\
\dot{w} &= qu - pv + g\cos\theta\cos\phi + \frac{f_z}{m} \\
\dot{\phi} &= p + \tan\theta(q\sin\phi + r\cos\phi) \\
\dot{\theta} &= q\cos\theta - r\sin\phi \\
\dot{\psi} &= \frac{q\sin\phi + r\cos\phi}{\cos\theta} \\
\dot{p} &= c_1 pq - c_2 rq + L \\
\dot{q} &= c_5 pr - c_6(p^2 - r^2) + M \\
\dot{r} &= (c_8 p + c_1 r)q + N \\
\dot{N} &= (\cos\theta\cos\psi)u + (\sin phi\sin\theta\cos\psi - \cos\phi\sin psi)v + \\
&\quad (\cos phi\sin\theta\cos\psi + \sin\phi\sin\psi)w \\
\dot{E} &= (\cos\theta\sin\psi)u + (\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi)v + \\
&\quad (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)w \\
\dot{D} &= (\sin\theta)u + (\sin\phi\cos\theta)v + (\cos\phi\cos\theta)w
\end{aligned}
\tag{2.12}
$$

Using the inertia terms derived in [7]

$$
\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{bmatrix} = \begin{bmatrix} \frac{(J_{xz}(J_x - J_y + J_z)}{\tau} \\ \frac{J_z(J_z - J_y) + J_{xz}^2}{\tau} \\ \frac{J_z}{\tau} \\ \frac{J_{xz}}{\tau} \\ \frac{J_z - J_x}{J_y} \\ \frac{J_{xz}}{J_y} \\ \frac{(J_x - J_y)J_x + J_{xz}^2}{\tau} \\ \frac{J_x}{\tau} \end{bmatrix} \tag{2.13}
$$

$$
\tau = J_x J_z - J_{xz}^2 \tag{2.14}
$$

and the forces and moments

$$
\begin{aligned}
f_x &= \frac{\rho V_a^2 S}{2}\left[C_x(\alpha) + C_{X_q}(\alpha)\frac{cq}{2V_a^2} + C_{X_{\delta_e}}\right] + \frac{\rho S_{prop} C_{prop}}{2}\left[(k_{motor}\delta_t)^2 - V_a^2\right] \\
f_y &= \frac{\rho V_a^2 S}{2}\left[C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{br}{2V_a} + C_{Y_r}\frac{br}{2V_a} + C_{Y_{\delta_a}}\delta_a\right] \\
f_z &= \frac{\rho V_a^2 S}{2}\left[C_Z(\alpha) + C_{Z_q}(\alpha)\frac{cq}{2V_a} + C_{Z_{\delta_e}}(\alpha)\delta_e\right] \\
L &= \frac{1}{2}\rho V_a^2 S b\left[C_{p_0} + C_{p_\beta}\beta + C_{p_p}\frac{br}{2V_a} + C_{p_r}\frac{br}{2V_a}\right] \\
M &= \frac{\rho V_a^2 S c}{2J_y}\left[C_m(\alpha) + C_{m_q}(\alpha)\frac{cq}{2V_a} + C_{m_{\delta_e}}(\alpha)\delta_e\right] \\
N &= \frac{1}{2}\rho V_a^2 S b\left[C_{r_0} + C_{r_\beta}\beta + C_{r_p}\frac{br}{2V_a} + C_{r_r}\frac{br}{2V_a} + C_{r_{\delta_a}}\delta_a\right]
\end{aligned}
\tag{2.15}
$$

Where V_a is the airspeed, ρ is the air mass density, S_{prop} the effective surface of the propellers, S the surface of the airfoils, α is the angle of attack between the forward direction pitch and the body frame pitch. β is the sideslip between the forward movement direction and body frame yaw. All C values are aerodynamic constants for the specific aircraft found by experimentation. The values used in the simulations and controls are for the X8 craft, and the values are discussed in [16]. They are listed in appendix A.

2.7 Linearized equations of motion

A linearized model is found in order to increase the computational efficiency of the MPC. This is discussed in more detail in section 5.3.5. The system should be transformed to the form

$$
\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\
\mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}w
\end{aligned}
\tag{2.16}
$$

Linearization about the solution $\mathbf{y}^*$ is based on defining the system output and states as $\mathbf{y} = \mathbf{y}^* + \overline{\mathbf{y}}$, $\mathbf{x} = \mathbf{x}^* + \overline{\mathbf{x}}$ and using the Taylor series to define [12]

$$
\dot{\mathbf{y}}^* + \dot{\overline{\mathbf{y}}} = \mathbf{f}(\mathbf{y}^*, t) + \left.\frac{\partial \mathbf{f}(\mathbf{y}, t)}{\partial \mathbf{y}}\right|_{\mathbf{y}=\mathbf{y}^*} \tag{2.17}
$$

The Jacobian **J** becomes

$$\mathbf{J} = \left.\frac{\partial \mathbf{f}(\mathbf{y}, t)}{\partial \mathbf{y}}\right|_{\mathbf{y}=\mathbf{y}^*} \tag{2.18}$$

And we get the linearized definition of $\dot{\bar{\mathbf{y}}}$ as

$$\dot{\bar{\mathbf{y}}} = \mathbf{J}\bar{\mathbf{y}} \tag{2.19}$$

This gives us the system in the form of 2.16 by the set of equations shown bellow

$$\begin{aligned}
\mathbf{A}(t) &= \begin{bmatrix} \nabla_x f_1(x_0, u_0, t) \\ \vdots \\ \nabla_x f_n(x_0, u_0, t) \end{bmatrix} \\
\mathbf{B}(t) &= \begin{bmatrix} \nabla_u f_1(x_0, u_0, t) \\ \vdots \\ \nabla_u f_n(x_0, u_0, t) \end{bmatrix} \\
\mathbf{C}(t) &= \begin{bmatrix} \nabla_x g_1(x_0, u_0, t) \\ \vdots \\ \nabla_x g_n(x_0, u_0, t) \end{bmatrix} \\
\mathbf{D}(t) &= \begin{bmatrix} \nabla_u g_1(x_0, u_0, t) \\ \vdots \\ \nabla_u g_n(x_0, u_0, t) \end{bmatrix}
\end{aligned} \tag{2.20}$$

Where

$$\nabla_x f(x_0, u_0, t) = \begin{bmatrix} \frac{\delta f(x_0,u_0,t)}{\delta x_1(t)} & \frac{\delta f(x_0,u_0,t)}{\delta x_2(t)} & \dots & \frac{\delta f(x_0,u_0,t)}{\delta x_n(t)} \end{bmatrix} \tag{2.21}$$

However, as we assume fairly accurate state feedback, **y** is set as equal to **x**. Therefore C and D is not considered.

This process is done in [7] to achieve a linearized model for small unmanned aerial vehicles with decoupled lateral and longitudinal components. This is the model that is used for the MPC in section 5.3.5. The resulting linear equations of motion will be presented, but the reader is refered to [7] for the complete spelling of the calculations. Note that the control signal δ_s is modelled as δ_e and δ_d as δ_a. The equations for all constants are given in appendix A

Lateral:

$$\begin{bmatrix} \dot{\bar{v}} \\ \dot{\bar{p}} \\ \dot{\bar{r}} \\ \dot{\bar{\phi}} \\ \dot{\bar{\psi}} \end{bmatrix} = \begin{bmatrix} Y_v & Y_p & Y_r & g\cos\theta^*\cos\phi^* & 0 \\ L_v & L_p & L_r & 0 & 0 \\ N_v & N_p & N_r & 0 & 0 \\ 0 & 1 & \cos\phi^*\tan\theta^* & q^*\cos\phi^*\tan\theta^* - r^*\sin\phi^*\tan\theta^* & 0 \\ 0 & 0 & \cos\phi^*\sec\theta^* & q^*\cos\phi^*\sec\theta^* - r^*\sin\phi^*\tan\theta^* & 0 \end{bmatrix} \begin{bmatrix} \bar{v} \\ \bar{p} \\ \bar{r} \\ \bar{\phi} \\ \bar{\psi} \end{bmatrix} + \begin{bmatrix} Y_{\delta_a} \\ L_{\delta_a} \\ N_{\delta_a} \\ 0 \\ 0 \end{bmatrix} \bar{\delta_a} \tag{2.22}$$

Longitudinal:

$$\begin{bmatrix} \dot{\bar{u}} \\ \dot{\bar{w}} \\ \dot{\bar{q}} \\ \dot{\bar{\theta}} \end{bmatrix} = \begin{bmatrix} X_u & X_w & X_q & -g\cos\theta^* \\ Z_u & Z_w & Z_q & -g\cos\theta^* \\ M_u & M_w & M_q & -g\cos\theta^* \\ 0 & 0 & 1 & 0 \\ \sin\theta^* - \cos\theta^* & 0 & u^*\cos\theta^* + w^*\sin\theta^* & 0 \end{bmatrix} \begin{bmatrix} \bar{u} \\ \bar{w} \\ \bar{q} \\ \bar{\theta} \end{bmatrix} + \begin{bmatrix} X_{\delta_e} & Y_{\delta_t} \\ Z_{\delta_e} & 0 \\ M_{\delta_e} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \bar{\delta_e} \\ \bar{\delta_t} \end{bmatrix} \tag{2.23}$$

2.7.1 Trim conditions

[25] suggests a set of systems linearized about different flight conditions, and a algorithm who switches these models appropriately. In this text however, one model will be used, linearized about the UAV trim conditions and the controller will constrain the angular states to be reasonably close to trim. The trim condition is given when the UAV is in a steady state having no acceleration. The Matlab tool developed by *Kristoffer Grye* [16] is used to get these values for the X8 UAV. The tool uses the method described in [3] and [7], and in the interest of brevity, the reader is referred to these works for the full presentation of how to find the trim conditions. The trim condition values for the X8 flying-wing UAV is presented in A.

CHAPTER 3

Concepts and terminology of deliberative control by EUROPA and T-REX

This chapter is rather short, but it was deemed necessary to keep as the concepts and terminology explained here will be key for most of the other chapters. Here the reader is introduced to ”the way of thinking” when dealing with domain modelling and dividing a system hierarchy as in EUROPA and T-REX.

3.1 Terminology of T-REX and Europa

When deliberating a plan there must be some way of defining the **domain** in which the plan is performed. The domain consists of all the possible outcomes of the plan, and following the EUROPA concepts in [1], this is done by constraining what, when and how actions are possible to place in the plan. All actions or states in this plan is called **predicates**. An object can have several possible predicates. It can for example be at a location, be going towards a location or gather a sample at that location. When placing such a predicate in a specific time interval in the plan, we place what is called a **token**. Tokens are instances of a predicate with **temporal attributes**, defining the interval it can start and end in. The attributes can both be **singleton** values holding a single time instance, or it can define an interval. Say we want to be at a specific place by the end of the day. This is a **goal**, with the temporal constraint of being ”while” the day ends. In order to achieve this goal we can start walking at any time

that day, as long as the duration of the walk don't make it too late. When looking at the different predicates mentioned above, it is clear that no one can both be at a place, and be walking to it at the same time. This can be ensured be declaring the predicates in a **timeline**. In a time line only one token can exist in the same time window. A timeline with tokens are illustrated in figure 5.1.

When creating the AMPE system, The plan will consist of several such timelines. Every state of the UAV to be defined in chapter 2 is a separate timeline for every UAV, which can only have one value at any time. When executing the plan, the system moves along these timelines in real time, and the current states is controlled with the current planned value as a reference. The real-time functionality of online planning and dispatching of goals on these timelines, while moving the current reference is what T-REX implements and facilitates [22], [26].

Throughout this thesis, the terms **module** and **teleo-reactor** (or just **reactor**) will be separated. The module will define the functionality and the tools used to implement a planning level in the AMPE system. These levels are the crude path planner, the "scientist" also named refined path planner, and the "navigator". The reactor will refer to the T-REX reactor encapsulating the module, placing it in the larger system with the required functions for parallelisation, synchronization and "translation" and flow of system signals between the modules.

CHAPTER 4

The AMPE system implementation

In this chapter, the entire AMPE system is illustrated with the With its agents, modules and timelines. Some terms and concepts are restated in order to present a complete reference system overview. These concepts is stated in section 4.2.1 which can be skipped if the T-REX and EUROPA terminology is understood.

First it is explained how the system is divided into a hierarchical parallel system by using T-REX, then the complete system is shown. Each reactor and module functionality is detailed and the program flow is shown in section 4.3. The next chapter will then explain the methods used by each module in order to implement the said functionality.

4.1 Modularizing the system for hierarchical synchronous planning and execution using T-REX

The T-REX agent is built as a program control unit with several reactors being responsible for the different tasks of the system Each reactor can provide internal timelines and subscribe to external timelines. When providing an internal timeline the reactor can post observations and are given requests when other reactors posts goals to this timeline. When a reactor subscribes to an external timeline, it can post goals to it and receives notifications when an observation is posted by another reactor. This process is illustrated in figure 4.1.

[26]

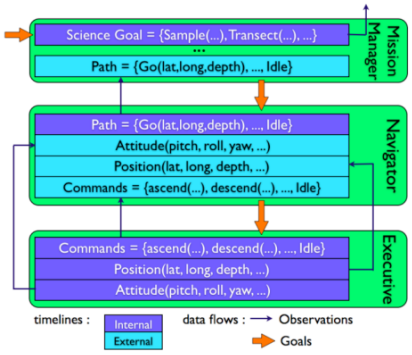


Figure 4.1: [22]

First, the T-REX agent reads a configuration file where all reactors, and their respective data, is defined. Then the respective reactors are constructed, which in turn construct the objects that hold the modules contained in each reactor. Then the planning and execution loop is started. For each **tick** the agent checks whether the reactor has work to do, meaning some deliberation process is needed. If so, a function is run letting the reactor check in on the deliberation process of the module. finally, the agent **synchronizes** all reactors. If a module is done deliberating, it will dispatch the new goals planned by the process, and all reactors will receive goals if they own the appropriate timelines. If some state handled by a reactor is changes, a observation will be dispatched on the appropriate internal timeline, and all reactors subscribing to this timeline will be notified. Figure 4.1 illustrates the program flow of the agent ticks. The left illustration shows how a tick is defined between synchronizations, and the deliberation in the course of this tick. The right side of the figure shows hoe goals and observations are dispatched at the synchronization. T-REX can also define a reactor latency which allows the reactor to deliberate over several ticks.

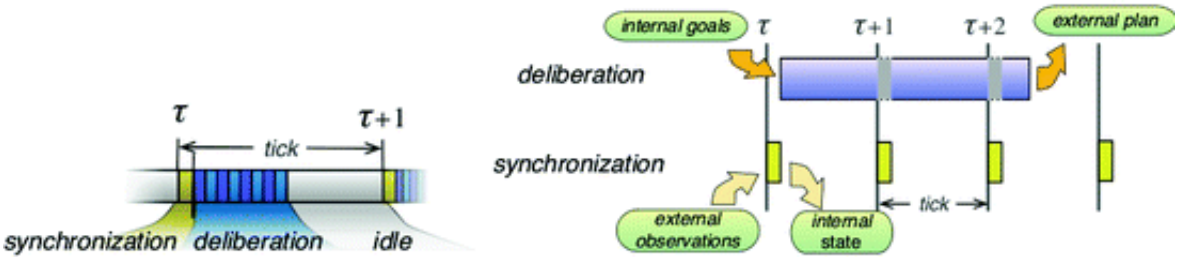


Figure 4.2: The figure illustrates the course of a tick on the left side, and how goals and observations are dispatched at the right side [26]

The AMPE system have four different reactors, one for each module and a **navigator** reactor which are responsible for

In order to select one of these reactors for the crude path planning, two different configuration files is created. One which defines the system with an EUROPA crude path reactor and one defining a MILP reactor. The rest of this chapter will present the methods and implementation of each of these modules. The next chapter will the present the system in it's entirety, including the agents, reactors and modules used, and how they are structured.

4.2 The AMPE system

This section will present the complete system with all timelines, reactors and modules and then explains the components one by one. Figure 4.3 illustrates the complete AMPE system.

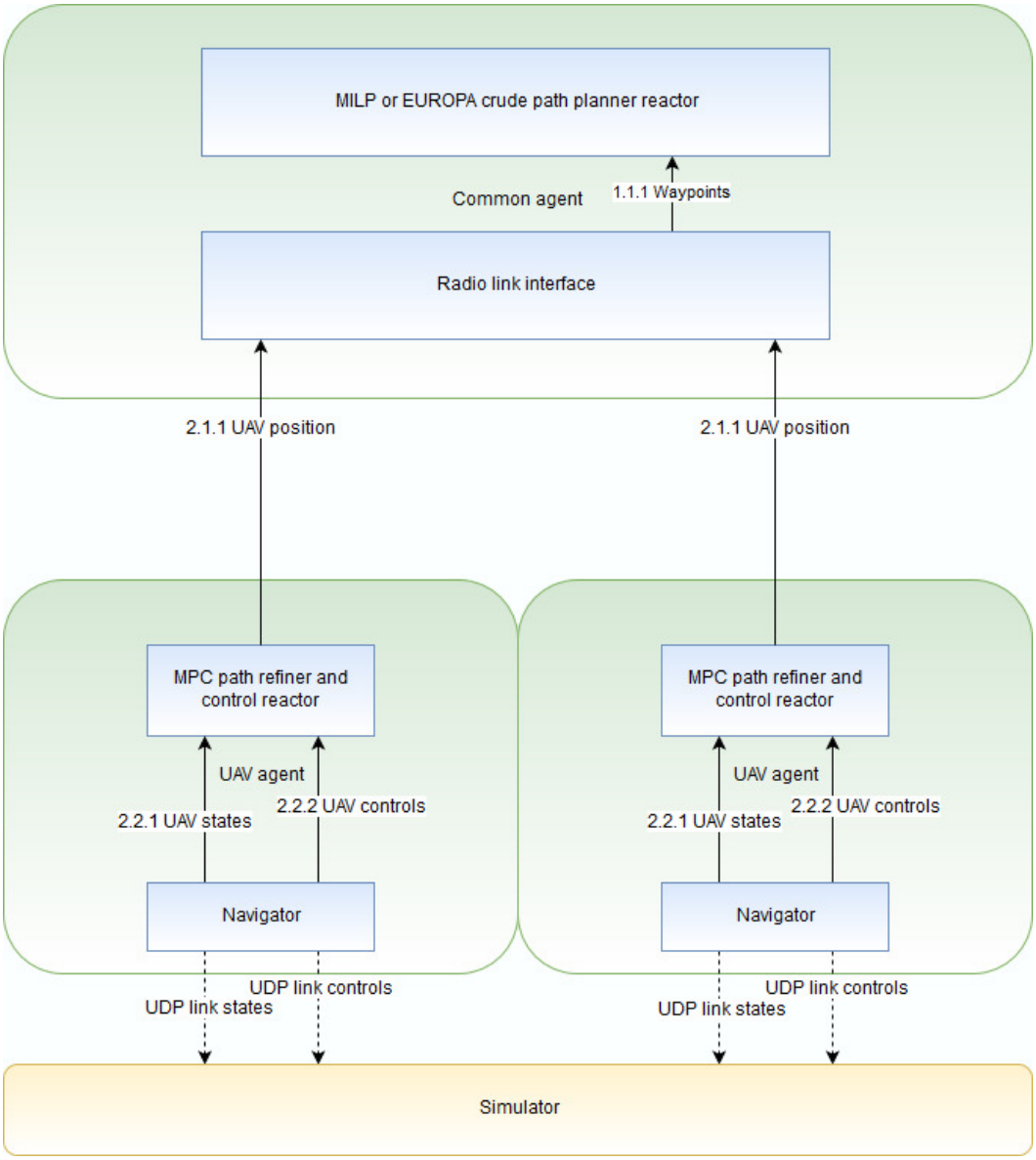


Figure 4.3: The complete system block diagram. The next sections will explain in detain the different parts of the figure.

4.2.1 System definitions

Agent

An agent is a complete encapsulation of some separated system. This is a T-REX ”program” and there will be two different agents in the system. The first is the common agent, which handles the planning of all UAV paths. The second is the UAV agent, which handles all local planning and execution aboard the UAV. It is important to note that one common agent can connect to several UAV agents. Both of these are colored green.

Teleo-reactor

A Teleo-reactor, or simply reactor, is a module inside the agent performing some specific task in the planning pipe-line. These are colored blue. The common agent consists of one reactor, the MILP path planner, and the UAV agent consists of the path processor, the EUROPA resource planner and the MPC controller.

Module

The module is the object holding the deliberation functionality inside each reactor. This also goes for the navigator module, which provides a UDP link to the simulator or physical UAV.

Timelines

A timeline is internal to the T-REX structure and are represented as solid lines and maintain both a current state, and goals with different time intervals. A timeline is internal to a reactor if the reactor is handling the current state and are receiving goals. A reactor can also subscribe to a timeline internal to another reactor in order to post goals on it. This makes it an external timeline to the subscribing reactor. The individual timelines will be specified in section 4.2.3.

4.2.2 Agents and Teleo-reactors

The common agent

The common agent is a of-UAV agent, that is responsible for the overall path planning of all UAVs. As the UAVs are by law required to uptain a communication link throughout the entire mission, it is assumed that this agent are constantly able to communicate the paths and states with the UAVs. The requirement of a common path planning comes from the optimization problem of creating a best possible way for all nodes to be visited by a UAV, minimizing all UAVs traveling time.

MILP crude path planner reactor

This is the reactor that creates the UAV paths in the common agent. It iterates with the biggest delay in the system and is seen as the top level in the planning problem. It sends the individual UAV paths to the correct vehicle by posting the waypoints as goals on the location timeline. Every path to a node is sent as a timestamped goal with the intermediate path as attached data to the goal. Tuning of path loss constraints must be carefully adjusted to make

sure nodes can not move out of range from a waypoint during the computing time of this reactor.

The crude path planner reactor is the highest level in the planning deliberation. It contains and runs the crude path planner which are responsible of planning the waypoints for all the UAVs, ensuring cooperation towards a time optimal plan. The MILP reactor does so with the use of the MILP crude path planner module. This uses mathematical optimization to achieve the plan, and is run at a certain interval by the reactor. The interval is ensured big enough that the module completes the plan in time for the next iteration. The reactor is responsible for extracting the plan, translate the waypoints to tokens and place them on their respective timelines as goals. It must also record observations on these timelines, and updates the data storage, position and resource status of the UAVs in the module before the planner is started.

EUROPA crude path planner reactor

The EUROPA reactor implements the same functionality as the MILP reactor, but uses an EUROPA module implemented by *NASA* with an interface developed by *Fredérik Py et.al.*. Goals are dispatched and observations updated as in the MILP reactor.

The UAV agent

This is the planning and controlling system aboard an UAV. It receives a crude path from the common agent, and follows this in an optimal manner by the use of MPC. All subsequent reactors belongs to this agent.

Navigator reactor

This is the reactor that as an interface towards the actual UAV actuators and sensors, or a simulator. In EUROPA and T-REX terminology, it is the implementation of the current token of the actual UAV states and controls. Goals dispatched to this reactor acts as control inputs to the UAV, and the observations posted by this reactor implements the feedback loop.

MPC path refining and control reactor

In this reactor, a Model Predictive Control is constantly iterating to generate optimal reference trajectory for a given time horizon. It uses state observations received from the navigator reactor to initialize the MPC module. Then it dispatches the control inputs to the navigator reactor timelines to act as reference control signals. Each reactor implements

some lookahead distance, meaning how far into the future the planner considers. For the crude path planners this is unnecessary as they considers the entire mission, but for the MPC reactor it defines the optimization horizon of the module.

4.2.3 Timelines and predicates

2.1.1 - Here does the first number indicates agent 2, the second number indicates reactor 1 in that agent, and the last number indicates the timeline. Timelines are illustrated in figure 4.3 as continuous lines.

Predicates used by all timelines

When planning the mission, the plan is formed as tokens placed on different timelines, as shown in section 3.1. These tokens are goals and the current state of the timelines of the vehicles are tokens placed as observations. These stems from predicates, and all timelines have a set of predicates from which i produces tokens.

1.1.1 Waypoints

This timeline is owned by the Radio links The waypoints timeline uses only a ”Waypoint” predicate giving a waypoint of all UAVs at a specific time.

2.1.1 UAV position

The UAV position timelines are owned by individual UAV path refiner reactors. These timelines uses the ”At” and ”Going” predicates, holding the NED coordinates as variables. ”Going” keeps the coordinates for where it’s going, and ”At” for where the UAV is positioned. ”Going” tokens are placed as goals from the Radio link interface to the appropriate timeline, and this works as waypoints. ”At” observations are used when a node is visited to remove the node from futher planning.

2.2.1 UAV states timeline and 2.2 UAV controls timeline

This timeline is owned by the Navigator reactor. The navigator produces ”State” tokens on this timeline as observations, which means that in the plan, this timeline represent the planned and current physical and complete state of the UAV. For the state of the controls, the same goes for the controls timeline. Control inputs are given as goals from the MPC path refiner reactor and the Navigator reactor will work to achieve these states and controls by communicating with the UAV systems.

UDP links

These links are not timelines but provide inter-process communication between the AMPE system and the UAV controller, or a simulator. The links are implemented using UDP with Boost asio, and this service will be described in section 6.2.4.

4.2.4 Implementation customized for local simulations

The system implementation for the simulations in part III, is somewhat simplified as there is no requirement for a radio link between the different agents. Therefore the system is implemented as a single agent, without the interface reactor for the common agent. The crude path planner reactor is responsible for the separation of the UAV timelines, and a UDP connection through localhost is used to communicate with the Simulink simulator made by *Kristoffer Gryte* in [16].

4.3 Flow diagram

The system proved several parallel services and a means of synchronizing them by T-REX. The system becomes somewhat complicated, and therefore a flow diagram will be presented to illustrate how the system shown in section 4.2 are implemented, and to show the execution flow of the agent. The diagram is illustrated in figure 4.4, and shows the agent and all reactors and modules. The control loop in the middle represents the agent, which are responsible for regularly synchronizing the reactors. The first thing that happens in the diagram is the deliberation command from the reactors. The reactors then translates all inter-reactor signals stored, like position or crude path, and starts the asynchronous deliberation of the modules. The agent then continues to synchronize the modules, and as shown in the figure, while the modules executes no plan is returned to the agent. However, as soon as the module completes the result is stored at the reactor, and at the next synchronization, the agent collects and dispatches that data. A new deliberation command is then sent, and the program loops all aforementioned tasks.

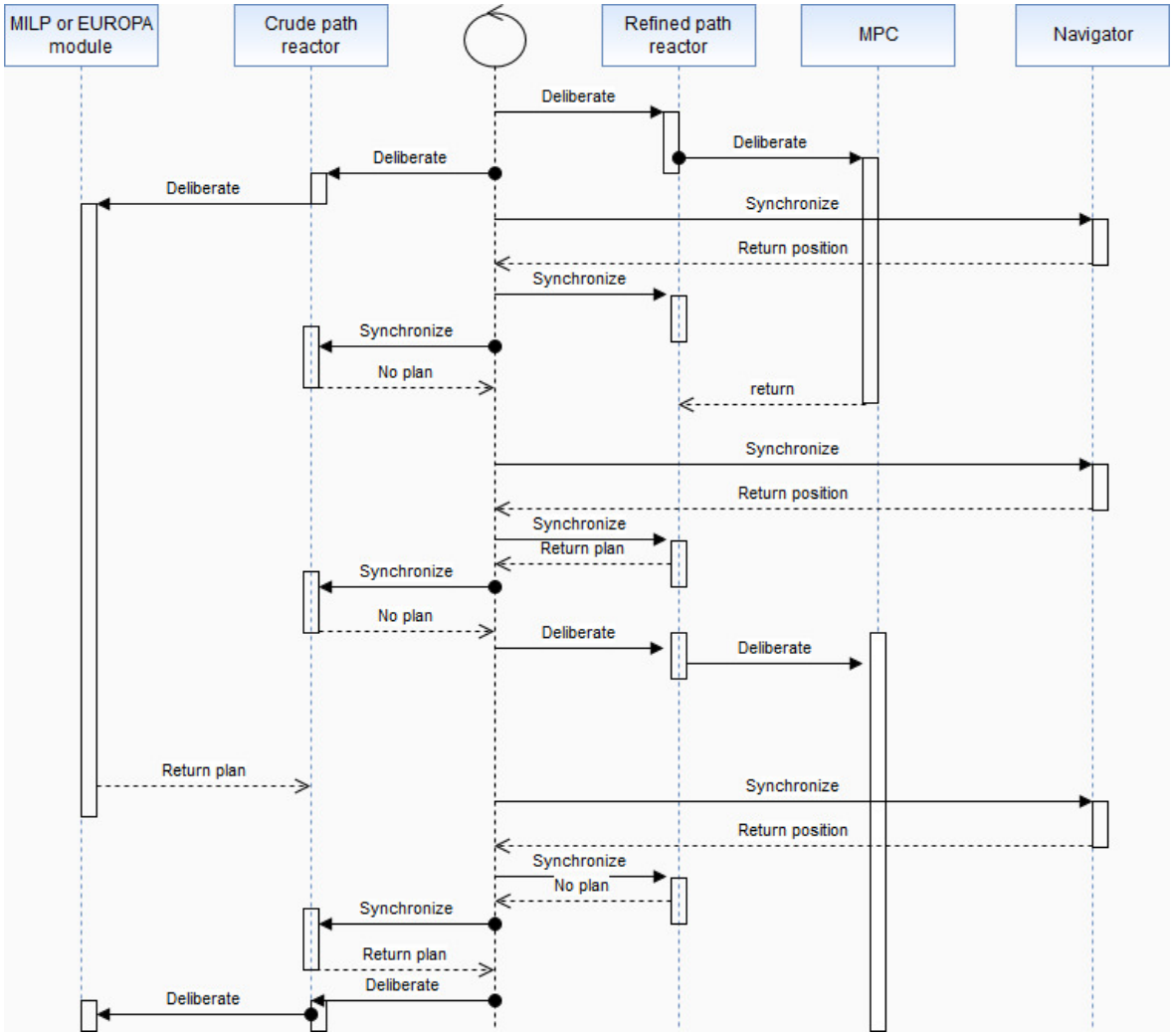


Figure 4.4: A flow diagram for the implemented agent program with all external modules. The control loop in the middle is the agent, responsible for synchronizing all reactors.

CHAPTER 5

The modules of the AMPE system

In this chapter, all modules are presented with their methods for implementing the functionality of which they are responsible. How EUROPA solves the planning problem is shown in section 5.1.1 before the nddl domain model used by EUROPA is explained. Then the concepts of mathematical optimization and Mixed Integer Linear Programming is shown. The MILP model used, is presented and explained before it is shown how AMPL is used to solve this program. Finally, the Model Predictive Control method is illustrated and models and constraints used is shown. It is also explained how ACADO toolbox is used to implement the MPC and how ACADO does so efficiently.

5.1 Planning a crude overall mission - The upper level module using EUROPA

In accordance with the intended design of T-REX [22] the crude path planner module is first implemented by an Europa reactor. Europa, introduced in section 1.8, offers the ”new domain description language” or nddl language (pronounced ”noodle”) [4], [1], [26]. This language is designed to describe the domain of the planning problem, and will be read by EUROPA and constraints the solution. The way this is modeled is somewhat similar to writing optimization programs as it is based on adding constraints to the feasible solutions. It is however built quite different, and in the next section the principles of nddl will be explained

as well as the architecture of EUROPA. Then the crude path model will be presented in section 5.1.1.

5.1.1 EUROPA and the new domain description language

The new domain description language is described in detail in the nddl reference manual [4] and the concepts used in the EUROPA module will be explained here. Nddl utilizes predicates, tokens and timelines as detailed in section 3.1. In addition, nddl defines **actions**. These are specializations of predicates, and the solver will search through the actions in the model in order to try and reach the goals of the problem. The actions will be given temporal attributes and behaves just like predicates once it is placed. The language also defines **rules**. All predicates and actions can be appended with rules, which constraints the placement of tokens of these predicates and actions with **conditions** and **effects**. Conditions tells the solver the requirements the plan must meet in order to place the token, and will consist of temporal relationship to other tokens. Say one wants to by a banana at the grocery store. A condition will be that during the buying process, the person must be at the store. After that, the person would own the banana. Here, ”buying” is the action and ”during” is the temporal relationship between the buy action and an ”at” token. ”At” is therefore a predicate.

Nddl is an object oriented language, and defines **classes** which can be instantiated as **objects**. The person and the banana as well as the store would be objects, and the buying action, ”at” predicate, and the ”own” predicate would be owned by the ”person” object. The ”own” predicate specifies that the person own a banana in this case. This would be contained by a **variable**. Both classes, predicates and actions can contain variables, and in this case a ”product” variable owned by the ”own” predicate would be specified as a banana. The last elements to mention is the **goals** and **facts**. Goals are specified after instantiating all objects. These are tokens with a temporal relation between each other, or a specific start and/or end time. Once tokens are created and all rules and temporal conditions are realized the plan is complete. Facts are specifications of tokens with variables that are in place when the solver is started.

Objects can be specialized as timelines. As objects can own predicates, this will give the object the behavior of a timeline, making the tokens mutually exclusive at time steps. The object can only have one token, at a time step. Object timelines with tokens and their respective rules are shown for the buying example in figure 5.1 All variables and classes have a domain in EUROPA. The domain of a class will be all objects created in the nddl, and variables like ”string” and ”float” can be restricted to certain ranges. This is what is done

when setting the start and end times of a token, as well as the duration. Since the temporal conditions of a goal most likely can be reached be starting our actions at different times, the time attributes of a token is restricted to earliest and latest start, and earliest and latest end. If these or the duration is specified, all these attributes will naturally constrain each other.

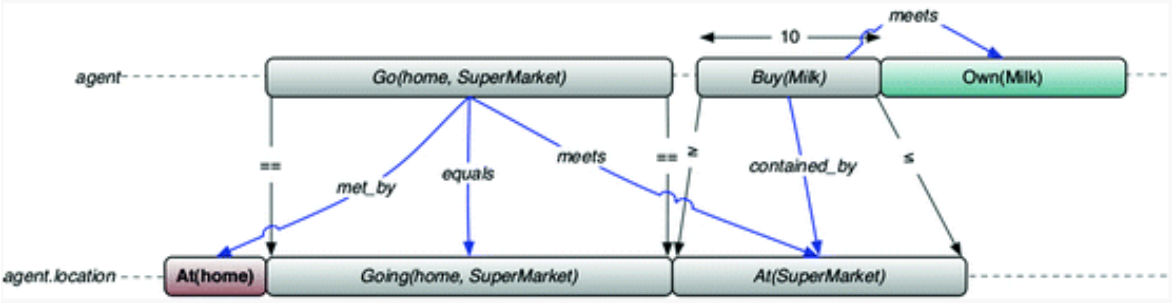


Figure 5.1: An EUROPA timeline for buying milk in a shopping domain [26].

The Europa planner architecture overview is shown in figure 5.2. When initiating a solver, a domain model must first be loaded. This will supply the rules engine and the constraints reasoning engine. The solver(s) will then try to add action tokens in order to reach the set goals. As the rules demands certain tokens to be in place at certain time, the rules engine will create **slave tokens** when placing a token. The solver will try to place these new tokens in accordance with their own rules, and if necessary, new slave tokens are created from these tokens and so on. As the solver iterates through these steps, the plan is added to the plan database. If the addition of a slave token creates a flaw in the plan, for example overstepping the constraints of a timeline object, this is captured by a handler which tries to remedy the flaw.

5.1.2 The nddl model formulation

The crude mission problem will be modelled as a nddl domain model in this section. When solving the problem using EUROPA, a plan with the latest and earliest times possible for all actions in order to achieve all goals are given. The solution to in some sense optimize the time horizon of the mission, for all tokens produced by the plan, only the early start times are used. This gives only singleton values for all temporal attributes and gives what EUROPA calls a **grounded plan** [1].

The domain is modelled around datagrams. Each node have data, and all nodes, UAVs and the base station can own such a datagram. When the solver is initialized each node is given some data, and the goals of the model is for the base station to own all datagrams by the

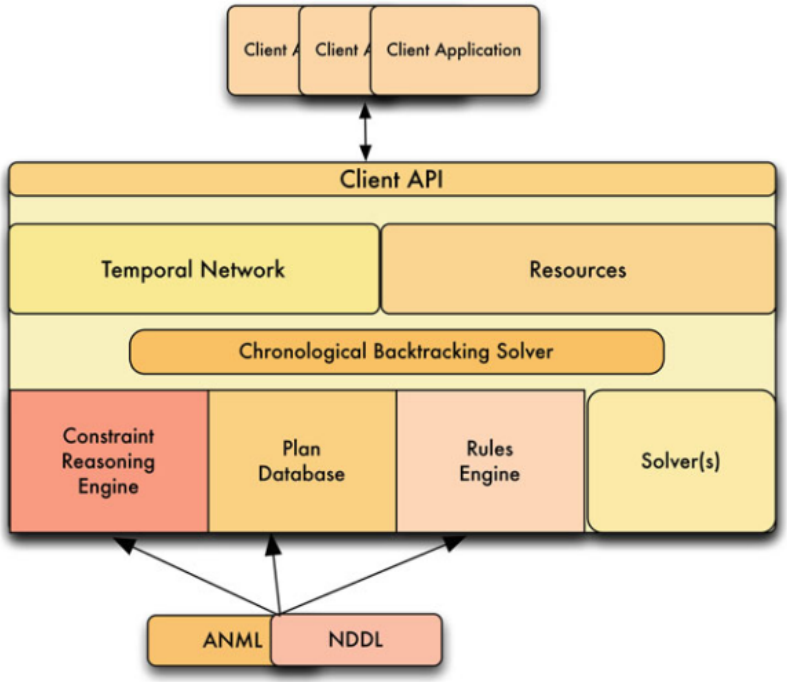


Figure 5.2: An overview of the EUROPA architecture [26]

end of the mission.In addition for the UAVs. the base station is the only data sink, which ensures the UAV is at the base area on mission completion. For the resource management some inspiration is taken from the official NASA Rover example [5].

```
class UAV;
class Navigator;
class Battery;
class Node;
class Data;
class DataContainerState;
class DataContainer;
class Location;
class BaseStation;
```

This is the list of all classes that will be defined in this model. The first class that are defined is the Location class. This contains the NED coordinates of the owning object. It will be used later in predicates to specify locational constraints.

```
class Location
```

```
{
        float n;
        float e;
        float d;
        Location(float _n, float _e, float _d)
        {
                n = _n;
                e = _e;
                d = _d;
        }
}
```

The next class is a Data class. Objects of Data will be used in predicates, telling some object that it is owning this specific data message. This way, each node can be given a unique message, and goals can be set to retrieve these to the base station.

```
class Data
{
        string name;
        //int size;

        Data(string _name)
        {
                name = _name;
        }
        Data()
        {
                name = "unspecified";
        }
}
```

The DataContainerState is used for the nest class which ill contain data. A data container can only have one state at a time, it can have data, be transmitting that data or it can be empty. All of these are predicates, and by inheriting the Timeline class by EUROPA. These predicates are guaranteed to be temporal exclusive.

```
class DataContainerState
        extends Timeline
```

```
{
        predicate HasData {}
        predicate IsTransmitting {}
        predicate IsEmpty {}
}
```

The TrancieverState class is like the DataContainerState but handles the base stations ability to send and receive data. It has the predicates allowing it to be free or transmitting.

```
class TranscieverState
    extends Timeline
{

        predicate FreePred {}
        predicate TransmittingPred {}
}
```

DataContainer class is responsible for holding the data of a node. It keeps a data object, which is the unique message that is held by the node, and a state timeline. It is implemented with several constructors, giving different ways of initializing an object.

```
class DataContainer
{
        Data data;

        DataContainerState state;

        DataContainer(string dataName)
        {
                data = new Data(dataName);
                state = new DataContainerState();
        }
        DataContainer()
        {
                data = new Data();
                state = new DataContainerState();
        }
```

```
        DataContainer(Data _data)
        {
                data = _data;
                state = new DataContainerState();
        }
}
```

The BaseStation class provides all functionality for the last node, where the UAVs are to end their mission. It keeps a state timeline for receiving capabilities of data, a location used by UAV GO predicates to be detailed soon, and a ReceivedDataPred predicate. As the base station is not a timeline, it can hold several such predicates. The predicate keep the variable data, which means that the predicate says that the base owns some data at some time.

```
class BaseStation
{
        Location location;

        TranscieverState state;

        BaseStation(float n, float e, float d)
        {
                location = new Location(n,e,d);
                state = new TranscieverState();
        }

        predicate RecievedDataPred
        {
                Data data;
        }
}
```

The navigator class is the class used externally when exporting the plan. The class specifies the location of an UAV at any time. It can be ”At” some location, or be ”Going” to from and to a location. The plan created by the model will consist of tokens of these two predicates. By extending the AgentTimeline class, all tokens on this timeline will be dispatched as goals, and all received observations will be used as a fact of which state token the timeline holds at some time.

```
class Navigator
        extends AgentTimeline
{




        predicate Going
        {
                Location from;
                Location to;
        }

        predicate At
        {
                Location at;
        }
}
```

The node implementation is specified bellow. It contains a location, and a data container. The data name variable is used for debugging. This data container is holding the datagram which the UAVs should collect.

```
class Node
{

        string dataName;
        Location location;

        DataContainer dataContainer;

        Node(string _dataName, float n, float e, float d)
        {
                dataName = _dataName;
                dataContainer = new DataContainer(_dataName);
                location = new Location(n, e, d);
        }
        Node(float n, float e, float d)
```

```
        {
                dataName = "unspecified";
                dataContainer = new DataContainer();
                location = new Location(n, e, d);
        }

        Node(Location _location, DataContainer _dataContainer)
        {
                dataName = "unspecified";

                dataContainer = _dataContainer;
                location = _location;
        }

}
```

The Battery class bellow implements resouce management to the model. It extends the Reservoir class which takes the initial value ic, the minimum value ll_min and the maximum value ll_max. This class is constrained to not have a resource value bellow ll_min, which is used by the UAV later.

```
class Battery
    extends Reservoir
{
    string profileType;

    Battery(float ic, float ll_min, float ll_max)
    {
            super(ic,ll_min,ll_max);
            profileType="IncrementalFlowProfile";
        }
}
```

This is the UAV class. It implements the predicates to own data, and the actions to GO to a location, a collect data action, and a send data to base action. These will be explained bellow.

```
class UAV
{
```

```
    Navigator navigator;
Battery battery;
    UAV()
    {
            navigator = new Navigator();
            battery = new Battery(1000, 0, 1000);
    }

    predicate HasData
    {
            Data data;
    }

    action GO
    {
            Location to;
    }

    action CollectData
    {
            Data data;
            Node possibleNodes;
    }

    action SendDataToBaseAct
    {
            Data data;
            BaseStation base;
    }



}
```

This is the action allowing the UAV to mode to an location. It first requires the navigator to be at some location, and stores that value. This is used as an effect to ensure that the duration of the action matches the distance between from and to. To is set by the solver, and after the action completes (after the duration) the navigator is set to be at this position. The last line of this action is to use a quantity of the battery equal to the distance travelled.

```
UAV::GO
{
        this.start >= 0;

        met_by(condition object.navigator.At _from);
        meets(effect object.navigator.At destination);
        eq(to, destination.at);

        equals(effect object.navigator.Going going);
        eq(going.from, _from.at);
        eq(going.to, destination.at);

        float dist;
        calcDistance(dist, _from.at.n, _from.at.e, destination.at.n, de
        duration <= dist;
        duration >= dist - 0.9999;
        start(effect Battery.consume tx);
        eq(tx.quantity, dist);
}
```

This action collects data from a node. It holds a possible nodes variable, which keeps all nodes as a domain, and require that the selected node have the data it want to collect. Then it require the UAV to be at the node position when collecting before setting the data container of the node to empty and gives the UAV an ”own” predicate with the data.

```
UAV::CollectData
{
        eq(duration, 2);

        this.start >= 0;
```

```
        possibleNodes.dataContainer.data == this.data;

        contained_by(condition object.navigator.At currentLocation);
        eq(currentLocation.at, possibleNodes.location);

        met_by(condition possibleNodes.dataContainer.state.HasData);

        starts(effect HasData inputData);
        eq(inputData.data, data);

        equals(effect possibleNodes.dataContainer.state.IsTransmitting);
        meets(effect possibleNodes.dataContainer.state.IsEmpty);

}
```

The send data to base action works similarly to the collect data predicate. It releases the own predicate of the UAV by the met_by constraint, and gives the base station the has data predicate. It locks the transceiver of the base station during the actions, which ensure no more than one UAV is sending at a time.

```
 UAV::SendDataToBaseAct
{
        duration == 2;

        met_by(condition HasData hasData);
        hasData.data == this.data;

        contained_by(condition object.navigator.At currentLocation);
        currentLocation.at == base.location;


        equals(effect base.state.TransmittingPred);
        meets(effect base.state.FreePred);

        starts(effect base.RecievedDataPred recieved);
        recieved.data == data;
```

}

5.2 Planning a crude mission path using mixed integer linear programming

5.2.1 Motivation and considerations

This will be the mathematical optimization alternative to the EUROPA module for the crude path planning. It utilizes Mixed Integer Linear Programming in order to create what EUROPA defines as a **grounded plan** [1], meaning all tokens time attributes are singleton values. It plans to be at a specific location at a specific time.

In this section the main principles of mathematical optimization is introduced. Then the idea and challenges of Mixed Integer Linear Programming is explained before it is used to model the path planning problem. An overview of this problem is given before the different part of the program is detailed further. The program used is proposed by Esten I. Grøtli and Tor A. Johansen in *"Motion- and communication-planning of unmanned aerial vehicles in delay tolerant network using mixed-integer linear programming"* [17], and in [18], and will be thoroughly explained in this section as it is a corner stone in the AI module replacement technique. Constraints for resource management will be added to this model.

Implemented with a receding time horizon, which gives it the functionality of a slow model predictive control. This concept will be

5.2.2 Convexity

Before moving on to the mathematical optimization concepts and methods, convexity must be defined. According to [23] a set is convex if for any two points A and B in the set, a straight line connecting them has every point inside the set. A function is convex if its domain is a convex set and if for any two points x and y in the set the following holds

$$f(ax + (1 - \alpha)y) \leq (1 - \alpha)f(y) \tag{5.1}$$

$\forall \alpha \in [0, 1]$ An optimization program is convex if the objective function is linear or quadratic, and all constraints are linear [23]. This is important because, if a program is convex, any local solution is a global solution, and this significantly simplifies the optimization. It is often not practically possible to solve a non-convex program if the problem is complex. Therefore the

rest of this chapter will go to some length to use integer variables to create a linear program. The integer variables however, makes the program non-convex, but there are fast solvers and algorithms in place to make solving mixed integer programs possible within some reasonable timeframe. The next section explains briefly why the program is non-convex, and the solver algorithms are touched upon in section 5.2.6.

5.2.3 Mixed Integer Linear Programing principles and formulation

In mathematical optimization you a set of variables is decided in order to minimize or maximize some objective function that defines optimality for your program. By minimizing an objective function that includes for example fuel consumption, you get a solution of variable values that gives the minimum fuel consumption. However, without further information, the solution would be trivial by setting the consumption to zero. Therefore we would want to constraint our solution to comply with physical laws as well as practical goals and issues. This is done by adding constraints to some or all variables that are to be optimized. These constraints forces the variables to be either greater or equal, smaller or equal, or equal to some constant or combination of other variables. For practical reasons we do not use less, and greater as the solution might be as close as possible to the constraint. The solver, finding a solution to the problem would then in the continuous case try to find the closest point to the constraint in a set of infinite points. The formulation of the program becomes

$$\begin{aligned} &minimize \quad f(\mathbf{X}) \\ &Subject\,to \\ &\qquad g(\mathbf{X}_i) \leq 0 : i \in I \\ &\qquad h(\mathbf{X}_i) = 0 : i \in \mathbb{E} \\ &\qquad \mathbf{X} \in \mathbb{R}^n \end{aligned} \tag{5.2}$$

Where is the objective function, a vector of the optimization variables of dimension n, I the set of inequalities, and E the set of equalities. Figure 5.3 illustrates such a program graphically with linear constraints and a quadratic objective. Namely a quadratic problem.

Mixed Integer Linear Programming generalizes this formulation co consider a mix of continuous and integer variables. There's some challenges in this formulation, where possibly the most prevalent is that the feasible set becomes non-convex when introducing integer variables. The feasible set is the set of solutions complying with the constraint, and when not convex the problem becomes np-hard. Still there is algorithms like branch and cut, that are

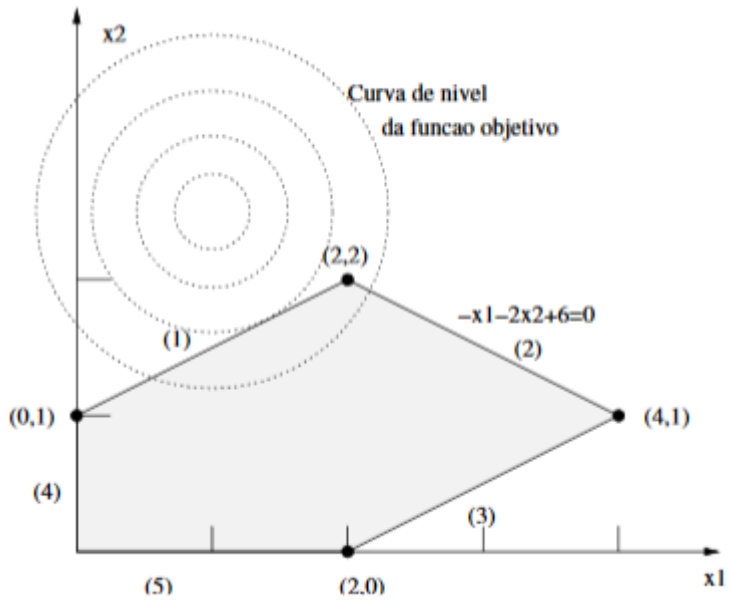


Figure 5.3: A graphic representation of a quadratic optimization program with two variables and three constraints. The greyed area is the feasible set.

able to solve the program efficiently. I will present the consept of this algorithm in some detail, and note that its is implemented in the solver CPLEX which will be used find the optimal solution to our final program. The MILP in canonical formulation becomes:

$$\begin{aligned} &maximize \quad c^T\mathbf{X} \\ &Subject\,to \\ &\qquad A\mathbf{X} \leq b \\ &\qquad \mathbf{X} = (\mathbf{X}_C, \mathbf{X}_I) \geq 0 \\ &\qquad \mathbf{X}_C \in \mathbb{R}^{n_C} \\ &\qquad \mathbf{X}_I \in \mathbb{Z}^{n_I} \end{aligned} \tag{5.3}$$

Where $\mathbb{Z}^{n_I}$ is the set of integer variables. The feasible set of such a program with two variables is illustrated in figure 5.5.

5.2.4 The MILP problem formulation

The program used was developed by *Esten Ingar Grøtli and Tor Arne Johansen* in [18], [11] and [17]. It is modeled as a time horizon N of time steps i. The UAVs will be denoted p in the set P_1^{np} where np is the amount of UAVs. In addition $(np+1)$ denotes the base station. Each

UAV should visit all waypoints w in the set T_1^W given during operations, where W is the amount of waypoints. The last waypoint represents the base station. Waypoints represents the nodes from which the UAVs should collect data, so nodes and waypoints will be used interchangeably to describe w. The program will model these data messages that is collected by the UAVs when visiting a waypoint, and the base station will operate as a sink for UAV to transfer the data. In the end the UAVs should return to the base station, and the program minimizes the time to do so after visiting all nodes. In addition the program should minimize the acceleration throughout the mission to avoid rapid fluctuations of speed. How this is modeled will be described in the sections to follow, and the objective function which should be minimized becomes

$$J = J_{acc} + J^{finnished} \tag{5.4}$$

Following the MILP model from [18], we can divide our model into sections. First we will constrain the solution by the UAV model, then the velocity will be approximated in a linear manner and constrained. The UAV acceleration will be modelled and weighted in our objective function in order to reduce actuation. We will formulate a bounding box of the UAV flight volume and constraint the path to this. We then constrain the model to accommodate all data transfer demands and processes. These are the data gathering process, the constraints of connectivity in order to transfer data, and how the data flow by delayed transmission. Some parts of the problem formulation of [17] is left out. In this article, anti grounding constraints are included. This is done by modelling the ground as a Triangular Irregular Network, an approximation of the ground shape as a linear combination of the corner points of triangles covering the ground. As our mission is simplified to a general location above open sea, these constraints falls under our definition of the flight volume and is therefore satisfied by constraining the UAVs to this volume. Throughout this section only, x and y and z are defined as the inertial frame axes.

UAV model

As the system only provides coarse path planning, a simple model is used for the UAVs. The program is

$$p_{p(i+1} = p_{pi} + \Delta t v_{pi} \tag{5.5}$$

$\forall p \in P_1^{np}, i \in I_1^N$ Where δt is the sampling time, and v_{pi} is the vector of the speeds along each axis of vehicle p at time i.

The approximation of euclidean distance

On several occasions throughout the coming model, there will necessary to constraint the distance $\sqrt{X^2+Y^2+Z^2}$. Doing this directly would render the program non-linear, and the distance must therefore be estimated in some way. In the article *"Low observability path planning for an unmanned air vehicle using mixed integer linear programming"*[11] a solution is prepared in the two dimensional case, and in *"Path Planning for UAVs Under Communication Constraints Using SPLAT! and MILP"* [18] the solution is expanded to the three dimensional case. By these solutions the distance from a center point, as a sphere with radius r, can be approximated by encapsulating the sphere with a polyhedron as illustrated in figure 5.4. The discretization level D^{vel} will decide the resolution of the polyhedron, which is then modeled as

$$\mathbf{v}^T \xi_{kl} \leq V \tag{5.6}$$

$$\alpha^{vel}\, \mathbf{v}^T \xi_{kl} \geq V - M_{kl}^{vel}(1 - b_{kl}^{vel}) \tag{5.7}$$

$k \in \{1, \dots, D^{vel}\}, l \in \{1, \dots, D^{vel}/2\}$

$$\sum_{k=1}^{D^{vel}} \sum_{l=1}^{D^{vel}/2} b_{kl}^{vel} = 1 \tag{5.8}$$

$\mathbf{v}$ is a vector, V is a scalar estimation of the length of v, α^{vel} is some value slightly smaller than 1, b_{kl}^{vel} is a binary optimization variable and χ_{kl} is given by

$$\xi_{kl} = \begin{bmatrix} \cos\theta_k \, \sin\phi_l \\ \sin\theta_k \, \sin\phi_l \\ \cos\phi_l \end{bmatrix} \tag{5.9}$$

Velocity

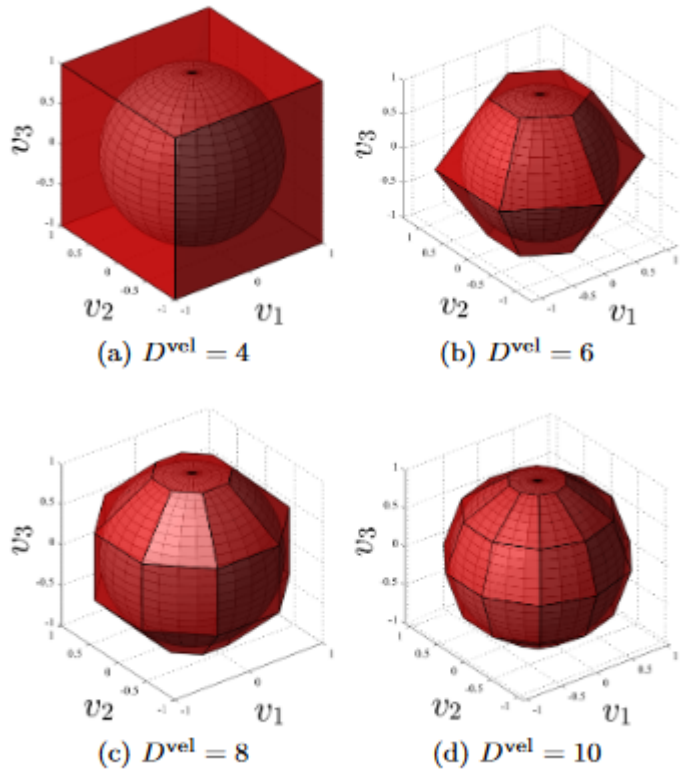
The velocity must be constrained between some minimum and maximum value, which can be done linearly by using the above method. The constraint of the estimated value for every time sample and every UAV the becomes

$$\mathbf{v}_{pi}^T \xi_{kl} \leq V_{pi} \tag{5.10}$$

$$\alpha^{vel}\, \mathbf{v}_{pi}^T \xi_{kl} \geq V_{pi} - M_{pkl}^{vel}(1 - b_{pikl}^{vel}) \tag{5.11}$$

$k \in \{1, \dots, D^{vel}\}, l \in \{1, \dots, D^{vel}/2\}\ \forall p, q \in P_1^{n_p}, i \in I_0^{N-1}$

$$\sum_{k=1}^{D^{vel}} \sum_{l=1}^{D^{vel}/2} b_{pikl}^{vel} = 1 \tag{5.12}$$



(a) $D^{\text{vel}} = 4$ (b) $D^{\text{vel}} = 6$

(c) $D^{\text{vel}} = 8$ (d) $D^{\text{vel}} = 10$

Figure 5.4: Illustration of the approximation of the euclidean distance. D^{vel} is the discretization level [18].

$\forall p, q \in P_1^{n_p}, i \in I_0^{N-1}$

Now, V_{pi} is the estimated velocities at time i and vehicle p. Setting $\underline{V}_p$ and $\overline{V}_p$ the minimum and maximum velocities of vehicle p respectably, velocity the constraint become

$$\underline{V}_p(1 - b_{piw}^{wp}) \leq V_{pi} \leq \overline{V}_p(1 - b_{piw}^{wp}) \tag{5.13}$$

The binary value b_{piw}^{wp}) will be visited in more detail in section 5.2.4, so for now it is sufficient to say that it will give zero at both ends when visiting the last waypoint, so the speed is forced to 0.

Position constraints

When operating a UAV one typically get a segregated airspace, and as such the UAVs should be constrained to this area. With the assumption of flying above open water, restricting the UAVs to a box a certain height above sea level will also prevent a crash landing. The

constraints simply become

$$\underline{x} \leq x_{pi} \leq \overline{x} \tag{5.14}$$

$$\underline{y} \leq y_{pi} \leq \overline{y} \tag{5.15}$$

$$\underline{z} \leq z_{pi} \leq \overline{z} \tag{5.16}$$

$$\tag{5.17}$$

$\forall p \in P_1^{n_p}, i \in I_1^N$ where the left and right inequalities represent min and max values respectably for each axis.

Acceleration

In order to keep the UAV thrust actuation within reasonably limits and prevent wear, acceleration is penalized by the following cost function

$$J^{acc} = \sum_{p \in P_1^{n_p}} \sum_{i \in I_0^{N-2}} \mathbf{r}_p^T \mathbf{w}_{pi}^{acc} \tag{5.18}$$

$$(v_{jpk} - v_{jpi}) \leq w_{pi}^{acc} \tag{5.19}$$

$$-(v_{jpk} - v_{jpi}) \leq w_{pi}^{acc} \tag{5.20}$$

$\forall p \in P_1^{n_p}, i \in I_0^{N-2}, k = i+1, j \in \{1,2,3\}$ $\mathbf{r}_p \in \mathbb{R}^3_{\geq 0}$ $\mathbf{w}_{pi}^{acc} = [w_{1pi}^{acc}, w_{2pi}^{acc}, w_{3pi}^{acc}]^T$ $\mathbf{r}_p^T$ is a non-negative weighing vector, and constraint 5.19 and 5.20 forces the elements of w to be the difference between the previous and current speed.

Anti-collision constraints

The anti collision constraint is simply modeled as a box around each UAV, having constraints on all axes on all UAVs at all times, to be a certain distance away.

$$dx - M_{pqi1} \leq x_{pi} - x_{qi} \leq M_{pqi2} - dx \tag{5.21}$$

$$dy - M_{pqi3} \leq y_{pi} - y_{qi} \leq M_{pqi4} - dy \tag{5.22}$$

$$dz - M_{pqi5} \leq z_{pi} - z_{qi} \leq M_{pqi6} - dz \tag{5.23}$$

$$\sum_{l \in 1..6} M_{pqil} \leq 5 \tag{5.24}$$

Task assignment

In order to determine whether a waypoint was visited, a bounding box will be modeled around all waypoints, and a binary variable b_{piw}^{wp} will denote if UAV p is at waypoint w at time i. The constraints are

$$\begin{aligned} x_{pi} - x^{wp} - d^{wp} &\leq M_{pw1}^{wp}(1 - b_{piw}^{wp}) \\ -x_{pi} + x^{wp} - d^{wp} &\leq M_{pw2}^{wp}(1 - b_{piw}^{wp}) \\ y_{pi} - y^{wp} - d^{wp} &\leq M_{pw3}^{wp}(1 - b_{piw}^{wp}) \\ -y_{pi} + y^{wp} - d^{wp} &\leq M_{pw4}^{wp}(1 - b_{piw}^{wp}) \\ z_{pi} - z^{wp} - d^{wp} &\leq M_{pw5}^{wp}(1 - b_{piw}^{wp}) \\ -z_{pi} + z^{wp} - d^{wp} &\leq M_{pw6}^{wp}(1 - b_{piw}^{wp}) \end{aligned} \tag{5.25}$$

$\forall p \in P_1^{np}, i \in I_1^N, winT_1^W$. $M_{pw1..6}^{wp}$ is set to be larger than the cross-section of the area of operations, and d^{wp} is the distance from the waypoint when it is considered to be visited. The constraint

$$\sum_{p \in P_1^{np}} \sum_{i \in I_1^N} b^{wp} \tag{5.26}$$

$\forall w \in T_1^W$ ensures that all nodes are visited once and only once during the mission.

In order to minimize the time consumption of the mission, several variables are defined. $\theta_p^{finnish}$ represents the time elapsed before UAV p returns to the base station by

$$\begin{aligned} \theta_p^{finnish} &\leq M^{finnish}(1 - b_{piW}^{wp}) + i b_{piW}^{wp} \\ \theta_p^{finnish} &\geq (i+1)(1 - b_{piW}^{wp}) \end{aligned} \tag{5.27}$$

$\forall p \in P_1^{np}, i \in I_1^N$. $M^{finnish}$ is selected as N. In order to mminimize the time horizon, the variable $\eta^{finnish}$ is constrained to be larger than $\theta_p^{finnish}$ and gives $J^{finnish}$ by

$$\eta^{finnish} \geq \theta_p^{finnish} \tag{5.28}$$

$$J^{finnish} = \gamma^{finnish} \eta^{finnish} \tag{5.29}$$

Where $\gamma^{finnish}$ is a scalar grater than zero, and $J^{finnish}$ is used in the objective function in section 5.2.4

Connectivity constraints

In order for a UAV to transmit data successfully at the specified data rate at an time instance is determined by constraining the UAV to be within a certain distance to the target. The distance R_{pqi}^{con} must be close enough for the radio propagation path loss to be within a certain limit. The radio propagation path loss can be estimated as $1/r^2$ where r is the distance, or done more accurately using a system named "SPLAT!". This uses real world geometry and calculates the local path loss. There is two options on constraining the actual distance against the feasible distance where one is by far superior. The first is to insert the distance as r in the radio path loss calculation. This would require us to estimate the path loss as a linear function through for example a CC-model, that being a linear combination function of points in a non-linear function. The other solution is to estimate the euclidean distance $\sqrt{x_{pi}^2 + y_{pi}^2 + z_{pi}^2}$ as a polyhedron following section 5.2.4. r is calculated as the threshold R_{pqi}^{con} and the relation

$$\begin{aligned} \tilde{b}_{pqi}^{con} &= 1 \\ &\Updownarrow \\ \chi_{pqi}^T \xi_{kl} - R_{pqi}^{con} &\leq 0 \end{aligned} \tag{5.30}$$

$\forall p, q \in P_1^{n_p+1}, k \in \{1, ..., D^{con}/2\}, l \in \{1, ..., D^{con}\}$
is enforced. For all nodes p and q at all time instances $\tilde{b}_{pqi}^{con}$ is one if node p can transmit successfully to node q. The constraints that ensures this relation is implemented following *"Control of systems integrating logic, dynamics, and constraints"* [8].

$$\chi_{pqi}^T \xi_{kl} - R_{pqi}^{con} \leq M^{con_\lambda}(1 - \tilde{\lambda}_{pqikl}^{con}) \tag{5.31}$$

$$\chi_{pqi}^T \xi_{kl} - R_{pqi}^{con} \geq \epsilon + (m^{con_\lambda} - \epsilon)\tilde{\lambda}_{pqikl}^{con} \tag{5.32}$$

$\forall p, q \in P_1^{n_p+1}, k \in \{1, ..., D^{con}/2\}, l \in \{1, ..., D^{con}\}$
where $M^{con_\lambda} = -m^{con_\lambda}$ is some value larger than the domain of $\chi_{pqi}^T \xi_{kl} - R_{pqi}^{con}$, and ϵ is some small number (typically the machine precision). λ_{pqikl} are boolean values that for every ξ_{kl} is true if $\chi_{pqi}^T \xi_{kl} - R_{pqi}^{con} \leq 0$ as in 5.30. $\tilde{b}_{pqi}^{con}$ represents whether or not UAV p is within range of UAV q, and should be true if all λ_{pqikl} are true. Using the same approach the constraint becomes

$$\begin{aligned} M^{con} - \sum_{k \in \{1,...,D^{con}/2\}} \sum_{l \in \{1,...,D^{con}\}} \lambda_{pqikl}^{con} &\leq M^{con}(1 - \tilde{b}_{pqi}^{con}) \\ M^{con} - \sum_{k \in \{1,...,D^{con}/2\}} \sum_{l \in \{1,...,D^{con}\}} \lambda_{pqikl}^{con} - M^{con} &\leq \epsilon + (m^{con} - \epsilon)\tilde{b}_{pqi}^{con} \end{aligned} \tag{5.33}$$

$\forall p, q \in P_1^{n_p+1}, k \in \{1, ..., D^{con}/2\}, l \in \{1, ..., D^{con}\}$
where $M^{con} = D^{con} * (D^{con}/2) - 1$. $\tilde{b}_{pqi}^{con}$ can only be true if the left-hand side is above or equal zero, and M^{con} ensures that the left-hand side will be zero if all λ_{pqikl}^{con} is 1 for vehicle p and q at time i. This is favorable as the solver do not have to consider the domain caused by the CC-model.

Data flow constraints for delayed transmission

When the UAV have gathered data upon visiting a sensor, it have several ways of propagating this data through the relay network. It can transmit directly and immediately to another vehicle or the base, and as we allow "ferrying" of data, it can buffer the data and physically transport it. This allows for a network with much larger range, but we require to keep track of the amount of data buffered so not to exceed the buffer size of the UAVs. [17] introduces several constraints to model these options in the program, which will be detailed here. The UAVs will be modelled as sources, creating a data message when visiting a node, and the base station will be the only sink. The rate of which data is gathered at nodes, or transmitted to the base station is c^{sensor}. The variable b^{sensor} will indicate when the UAV is gathering data, and is true when the UAV is servicing a task.

The messages will be modelled as m_{pisj} where p is the vehicle the message is currently on at time step i. The message will be identified by being created by vehicle s at timestep j. These messages can be divided into several smaller pieces, each being potentially routed differently back to the base station.

The data transmission rate between vehicles or the base station p and q at time step i for a message created at time step j by vehicle s, is denoted c_{pqisj}. These variables as well as the data message sizes can not be less than zero, which becomes the constraints

$$m_{pisj} \geq 0 \tag{5.34}$$

$\forall p \in P_1^{n_p+1}, j \in I_1^N, s \in P_1^{n_p}, i \in_j^N$

$$c_{pqisj} \geq 0 \tag{5.35}$$

$\forall p, q \in P_1^{n_p+1}, j \in I_1^N, s \in P_1^{n_p}, i \in_j^N$

Then the flow of data between nodes are modelled as

$$m_{pisj} = \Delta t \left(c^{sensor} b_{pj}^{sensor} - \sum_{q \in P_1^{n_p} \setminus \{p\}} c_{pqisj} \right) \tag{5.36}$$

$\forall s \in P_1^{n_p}, j \in I_1^N, i = j, p = s$

$$m_{pisj} = m_{p(i-1)sj} + \Delta t \left(\sum_{q \in P_1^{n_p} \setminus \{p\}} (c_{qpisj} - c_{pqisj}) \right) \tag{5.37}$$

$\forall p, s \in P_1^{n_p}, j \in I_1^N, i \in I_{j+1}^N$

These constraints can be thought of as the conservation of data. 5.36 represent the gathering of data where the first term on the right-hand side is the data stored on the buffer, and the subtraction of the second term is the data that is immediately transferred to another UAV. 5.37 ensures that the message at a time instance equals the message already stored on the vehicle, plus the received message data from other UAVs, minus the message data transferred to other UAVs.

Then the sum of message data on a vehicle is constrained to be less than the buffer size $\overline{h}_p$

$$\sum_{s \in P_1^{np}} \sum_{i \in I_1^i} m_{pisj} \leq \overline{h}_p \tag{5.38}$$

$\forall p \in P_1^{n_p}, i \in I_1^N$

The next constraint ensures that is a message created by a vehicle at a time instant is zero, it remains zero throughout the time horizon.

$$m_{pisj} \leq b_{pj}^{sensor} M^{msg} \tag{5.39}$$

$\forall p, s \in P_1^{n_p}, j \in I_1^N, i \in I_j^N$

The base station $(n_p + 1)$ can only receive data, and is constrained to

$$c_{(n_p+1)qisj} \leq 0 \tag{5.40}$$

$\forall q, s \in P_1^{n_p}, j \in I_1^N, i \in I_j^N$

We the constrain the transmission rate at time instance i of the message created by vehicle s at time instance j, between vehicle p and q, to be zero if they are out of range and less than a max transmission rate C^{max}

$$c_{pqisj} \leq C^{max}\tilde{b}^{con}_{pqi} \tag{5.41}$$

$\forall p, s \in P_1^{n_p}, p \in P_1^{n_p+1}, j \in I_1^N, i \in I_j^N$

where b^{con}_{pqi} indicates connectivity as shown in section 5.2.4 Lastly, the transmission rate is constrained bellow a separate max input and output rate.

$$\sum_{q \in P_1^{n_p+1} \setminus \{p\}} \sum_{s \in P_1^{n_p}} \sum_{j \in I_1^i} c^{pqisj} \leq C^{max_out} \tag{5.42}$$

$$\sum_{q \in P_1^{n_p+1} \setminus \{p\}} \sum_{s \in P_1^{n_p}} \sum_{j \in I_1^i} c^{qpisj} \leq C^{max_in} \tag{5.43}$$

5.2.5 Resources constraints

The model should take into consideration the limited resources aboard the UAVs. These resources are the UAV battery for the actuators, and the battery for the UAV transceivers. The controllers in the subsequent reactors will optimize the control input around the trim conditions of the UAV. The power usage of the propeller is therefore modelled as the usage when travelling at the trim airspeed multiplied with distance traveled and the power consumption pr meter traveled. The consumption of other actuators are considered negligible in comparison and is not considered. The constraint becomes

$$\sum_{i \in I_1^N} V_{pi} U^{prop} \leq B^{prop} \tag{5.44}$$

$\forall p \in P_1^{n_p}$,

Where the battery capacity is B^{prop} and U^{prop} is the power consumption for each meter traveled. The same concept is used for transceiver power. The transceiver battery capacity is B^{tran} and a certain amount of power U^{tran} is drawn each time instance the sensor is used. This gives the constraint

$$\sum_{i \in I_1^N} b^{sen}_{pi} U^{tran} \leq B^{tran} \tag{5.45}$$

$\forall p \in P_1^{n_p}$

5.2.6 Solver algorithms

The program defined in this section is linear, but also not convex. This is because of the non-convex nature of using integer domains in the program. The requirement of convexity is that one can draw a straight line between any two points inside the feasible area of a program. However, in figure 5.5 a integer domain is shown, and the feasible areas are only the points at which the grid values are integers, causing a set of disconnected regions. There is still efficient algorithms to solve the mixed integer programs, and the branch-and-cut algorithm are able to solve our program in a reasonable time. The algorithm actually is a combination of several others, and I will explain the concepts of these algorithms in the coming section before wrapping them together in the final Branch-and-Cut algorithm.

Simplex

This algorithm is based on tracing the constraints of a program in a methodical way until it have found some point with a negative gradient on both sides of the current point on a constraint. It does this by introducing a new constraint with a new slack variable forcing the program along the selected constraint when the slack variable is increasing. It does this until reaching a new constraint, and adds a second constraint in the same fashion. This is iterated until the solution is found, or a solution within some limit.

Branch-and-Bound algorithm

A relaxation of the problem is a simplification that in some way makes the optimization program easier to solve, giving a less optimal, and maybe infeasible solution to the original problem. The advantage of this is that we get a upper bound on our solution. The optimal solution will not be better than the solution to the relaxed problem. One such relaxation is the integer relaxation dropping the requirement of a integer solution. When acquiring such a solution one could simply floor the decimal solution, but this will not necessarily give the right solution as figure 5.5 illustrates. The top point is found after relaxation and the point straight below is found after rounding as the feasible set does not reach the above point. When using the convex hull however, the decrease after rounding causes the rightmost point to actually have the best objective value. The branch-and-bound algorithm works by splitting the problem into several sub-problems by the means of integer relaxation. It finds a solution, giving one or more of the integer variables a non-integer number. Then it splits the problem on all such variables, giving a problem with the constraint on the integer variable of being bellow the floor of the non-integer value, and another constrained to be above the ceiling

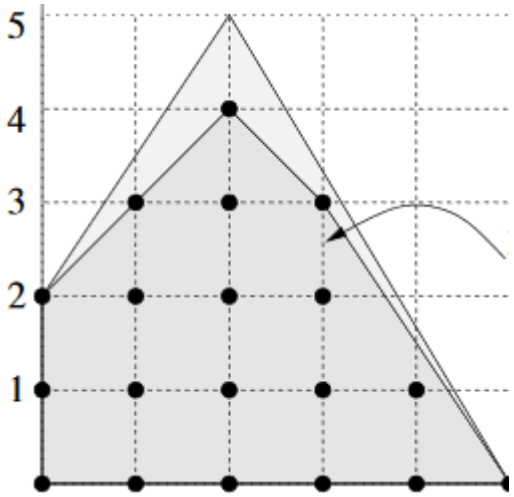


Figure 5.5: An illustration of the integer feasible set. NTNU.

of the value. This is repeated on the sub-problems until a integer solution is found on all integer variables. The best sub-problem solution is selected as the answer. To make this method viable and not likely to split the problem excessively, methods of pruning the sub-problem tree is used. Problems are not split if it is in-feasible, have fund a optimal solution, or have a relaxed solution (higher bound) lower than a current integer solution.

Cutting-plane algorithm

If a integer program has a convex hull, meaning the constraints have all ”corners” in integer solutions (illustrated with the inner boarder in figure 5.5) the integer relaxed solution will be optimal. The algorithm takes advantage of this property, and tries to create a convex hull. It uses a principle called Gomory-cuts, which will not be detailed here, but the main principle consists of using the simplex method to iteratively make cuts that tangents the actual convex hull to better and better fit the feasible set.

Branch-and-Cut algorithm

Branch-and-Cut is a combination of all of the above methods. It uses the Branch-and-Bound algorithm and then some iterations of the Cutting-plane algorithm for every node. Then the algorithm continues with the simplex method as the Branch-and-Bound algorithm. In practice, both the B-a-B and Cutting.plane algorithms can be fairly slow. This combination however tends to give good results.

5.2.7 Implementing The MILP crude path planner using AMPL, CPLEX and AMPLAPI

The program is implemented in C++ by using AMPL as a modeling interface towards the IBM solver CPLEX. AMPL is connected to the crude path module by the programming interface library AMPLAPI, from the same developers as AMPL. To illustrate and provide the MILP model written in the AMPL modeling language, the entire program is included in appendix C.

5.3 Refining a system dynamics dependent path using Model Predictive Control

5.3.1 Motivation

5.3.2 Motivation

In [27] we rely on being at waypoints at specific times. Also using dubins path would be a long way towards a refined path already. However, such path planning would have little consideration of the UAV dynamics. Therefore the MPC developed here is given more ”freedom” in sense of constraints and cost penalties of path choice. The MPC should minimize the distance from a UAV to the current waypoint at the appropriate time, and minimize fuel consumption, meaning thrust input, and minimize the changes of actuator states to prevent wear and tear.

5.3.3 Model predictive control principle

Following [13], model predictive control (MPC) is an algorithm for solving an optimal control problem (ocp). It does so by defining an optimization program that is constrained by the model of the vehicle that is to be controlled, and an objective function relating the states to a reference trajectory. The program defines both state and control variables for a certain steps into the future, and these are optimized according to the objective function. The first control output is then selected and passed to the actuator controls. The MPC then takes a new set of measurements and repeats this loop like shown in figure 5.6. In [13] a state estimator is implemented alongside the MPC, but as mentioned this is outside the scope of this report, and is therefore skipped.

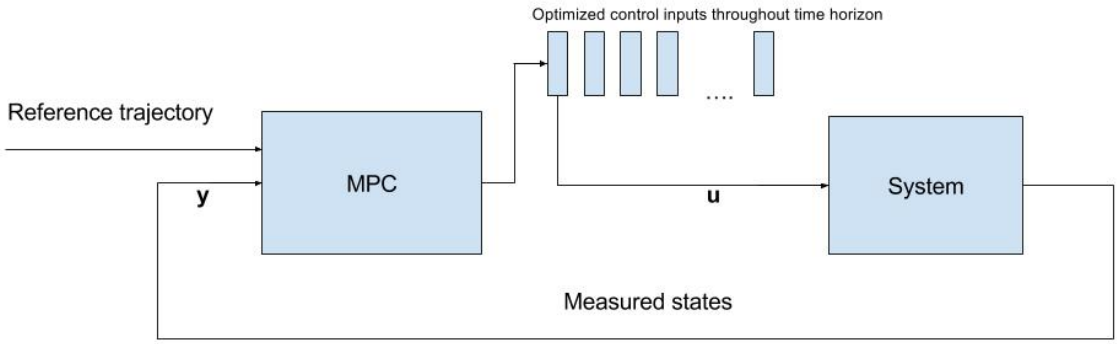


Figure 5.6: A diagram illustrating the execution flow of a MPC. The MPC takes a reference trajectory and the current state measurements and optimizes the control inputs throughout the time horizon. Then the first control input is selected and used. Only signals relevant to the controller without noise and disturbances is shown.

The optimization of a single control is shown in figure 5.7. The MPC will optimize the control from t to t+p, and this is defined as the time horizon or optimization horizon. The MPC follows the receding time horizon principle, meaning that as the time t moves, so does k, and the MPC optimizes over a constant interval p into the future. The figure shows how a predicted output converges on a reference trajectory using the predicted control input.

The algorithm of the MPC is defined in [13] as

Algorithm 1 Linear MPC with output feedback [13]

for t = 0,1,2,... **do**

Compute an estimate of the current state $\hat{x}_t$ based on the measured data up until time t

Solve convex QP problem (5.52) on the prediction horizon from t to t+N with $\hat{\mathbf{x}}$ as the initial condition.

Apply the first control move $\hat{\mathbf{u}}_t$

end for

5.3.4 Formulation

As the MPC optimizes over a time horizon at discrete time steps, the system should also be discrete as the MPC evaluates the system at discrete time steps. The system bellow is a nonlinear discrete system interpretable by the MPC.

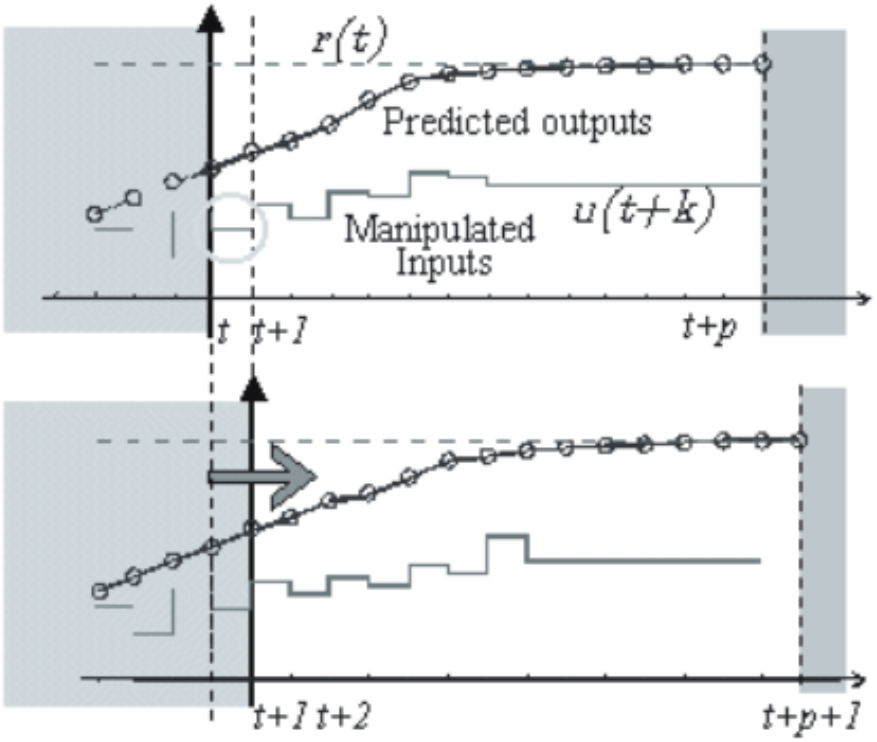


Figure 5.7: Example plot of a MPC iteration [25]

$$min \sum_{i=t..t+p} J_i \tag{5.46}$$

$$\begin{aligned} \mathbf{x}_{i+1} &= f(\mathbf{x}_i, \mathbf{u}_i) + v_i \\ \mathbf{y}_i &= h(\mathbf{x}_i, \mathbf{u}_i) + w_i \\ \mathbf{x}_0 &= given \\ \mathbf{x}^{low} \leq \mathbf{x}_i &\leq \mathbf{x}^{high} \\ \mathbf{u}^{low} \leq \mathbf{u}_i &\leq \mathbf{u}^{high} \\ \Delta\mathbf{u}^{low} \leq \Delta\mathbf{u}_i &\leq \Delta\mathbf{u}^{high} \end{aligned} \tag{5.47}$$

where **x** is the states, **y** is the measured outputs, **u** control vector and t is the current sample time. **v** and **w** is disturbances and noise respectively and we set these to 0 for the reminder of the chapter. $\Delta\mathbf{u}$ is the change in control input, and is constrained between a high and low value, along with the states and control inputs.

5.3.5 MPC for UAV control

A suitable model must be chosen to constraint the MPC to the UAV dynamics. The MPC is intended as an online control algorithm, and solving a complex program can be computationally expensive. Keeping an accurate UAV model without rendering the constraints nonlinear and thereby the program non-convex, becomes a challenge. However, by using the SQP algorithm, solving a nonlinear program is done quite efficiently in the ACADO library. This will be discussed in more detail in section 5.3.6. ACADO also allows for continuous models to be defined as constraints, and uses online shooting methods to get a discrete system [24] as in 5.47. Therefore the linear model presented in chapter 2 will be used in order to reduce the nonlinearities, and keep the program as close to quadratic as possible. However, a rotation is required to get an accurate position in the NED frame in order to use waypoints provided by the crude path planner, and this results in a nonlinear MPC. In order to avoid these nonlinearities the model would have to be simplified significantly in some similar way to the crude path program. As we want to adapt the crude path to the UAV model, keeping these simplifications is not suitable for the MPC. As such, using the linear UAV model and adding the non-linear rotations from body to NED frame has proven to be a good trade-off, by significantly reducing the computation time of using the nonlinear model 2.12.

5.3.6 The objective function and constraints

We want to control the states $\mathbf{x}$ along a reference trajectory γ^{ref} in some optimal way. The objective function will therefore include the absolute difference between the predicted states γ and the reference trajectory. We also want weight the usage of actuators in the optimization in order to reduce these, and as such limit fuel usage and actuators wear and tear. The ACADO MPC, to be detailed later, takes a least squares objective function, and as such the objective function presented in [13] is modified and these aspects are implemented in the objective function as

$$J = \int_{t=0}^{t+p} \frac{1}{2}(\gamma - \gamma^{ref})^T \mathbf{Q}(\gamma - \gamma^{ref}) + \frac{1}{2}\mathbf{u}^T\mathbf{R}\mathbf{u} + \frac{1}{2}\Delta\mathbf{u}^T\mathbf{R}\Delta\mathbf{u} \tag{5.48}$$

$$\min_{z\in\mathbb{R}^n} f(\mathbf{z}) = J \tag{5.49}$$

where $\mathbf{Q} \succeq 0$ and $\mathbf{R} \succ 0$ are weighting matrices, and $\mathbf{z} = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ...)$

In order to restrict the predicted states according to the system model, we need to add constraints. We also want to constraint the possible control input according to saturation. At last we want to constraint the possible predicted states to avoid too low or high speeds, or oddities like the UAV performing barrel rolls or loops. The full system including constraints take the form

$$\begin{aligned}
&\min_{z\in\mathbb{R}^n} f(\mathbf{z}) = J \\
&subject\,to \\
&\qquad \mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t \\
&\qquad x_0 = given \\
&\qquad \mathbf{x}^{low} \leq \mathbf{x}_t \leq \mathbf{x}^{high} \\
&\qquad \mathbf{u}^{low} \leq \mathbf{u}_t \leq \mathbf{u}^{high} \\
&\qquad \Delta\mathbf{u}^{low} \leq \Delta\mathbf{u}_t \leq \Delta\mathbf{u}^{high}
\end{aligned} \tag{5.50}$$

When adding constraints to the UAV states, the system might end up in a situation where the program becomes infeasible. Say, the MPC is started with a very high acceleration very close to the maximum allowed speed. This makes it impossible for the MPC to decelerate in time, and the problem will be infeasible. This is unacceptable as the control algorithm would fail, leaving the UAV uncontrollable. A solution is presented in [13] by adding the slack variables ϵ to the constraints and minimizing these variables in the objective function. We set

$$J_\epsilon = \rho^T\epsilon + \frac{1}{2}\epsilon^T\mathbf{S}\epsilon \tag{5.51}$$

And the new program becomes

$$\begin{aligned}
&\min_{z\in\mathbb{R}^n} f(\mathbf{z}) = J + J_\epsilon \\
&subject\,to \\
&\qquad \mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t \\
&\qquad x_0 = given \\
&\qquad \mathbf{x}^{low} - \epsilon \leq \mathbf{x}_t \leq \mathbf{x}^{high} + \epsilon \\
&\qquad \mathbf{u}^{low} \leq \mathbf{u}_t \leq \mathbf{u}^{high} \\
&\qquad \Delta\mathbf{u}^{low} \leq \Delta\mathbf{u}_t \leq \Delta\mathbf{u}^{high}
\end{aligned} \tag{5.52}$$

Where ρ and $\mathbf{S}$ are the weighting vector and matrix and

$$\epsilon \in \mathbb{R}^n \geq 0$$
$$\rho \in \mathbb{R}^n \geq 0$$
$$\mathbf{S} \in diag\{s, 1, ..., s_n\}, s_i \geq 0, i = \{1, ..., n_x\}$$

Using the controls of the X8, the control limits are presented in [16]. Using these limits, the MPC is constrained to the saturation of each control, being ± 0.3 for the ailerons and elevators, and between zero and one for the propeller. In addition, the attitude of the UAVs are constrained to be within reasonable limits of the trim condition of which the UAV is linearized about. This ensures the accuracy of the linear model, and restrict risky maneuvers such as loops. The roll and pitch are constrained to ± 30 degrees.

Implementation of MPC using ACADO

The MPC is implemented in C++ as a module of the AMPE system using ACADO Toolbox for optimal control. The toolbox provides a interface which is close to how the the program would be written mathematically. The expressions are stored symbolically allowing for the definitions of models accepted as constraints to the MPC. ACADO are able to solve non-linear programs and is designed for optimal control [19]. The toolbox accepts continuous functions by using shooting methods such as numerical Runge-Kutta methods to evaluate the model at discrete time steps, and implements fast solvers by the use of Sequential Quadratic Programming for nonlinear control. The main principle of these concepts will be visited briefly in the next two sections.

Runge-Kutta45

RK45 or Runge-Kutta4(5) is a numerical integrator method which allows for a specified accuracy in the estimation of the new state in the MPC. The full explanation of the method is provided in [12], but for the reader of this thesis it is sufficient to know that it uses the relationship of step length and the derivative of a function to estimate the function after that step length, and does so by dividing the step into several stages, using the derivatives at these intermediate points and the step length between them to decrease the error. When comparing method to a Taylor expansion series, it can be seen that a RK method is of a certain order, and the order of the error estimate can be read from the Taylor series matching the method. RK45 uses a 4. and 5. order numerical method to set the step length of the integrations in order to ensure a minimum error, a feature used as a standard by Matlab integrators [12].

SQP

Following [9], the Sequential Quadratic Programming, or SQP method used by ACADO [24] to solve nonlinear, non-convex programs, are based on iterativeliy finding a quadratic approximation to the objective function at each step of the MPC, and solve this relaxed problem. Linear approximations replaces the nonlinearities, effectively linearizing the function at each step about the predicted state. For each step, 2 SQP iterations have been found to be sufficient for the model used in this MPC.

5.4 The navigator module

This module provides the interface towards the vehicle, and in the AMPE system it represent the same as the navigator in the EUROPA model. When reactors are dispatching plans, this module is responsible for the position timeline of the UAV. Internally it communicates with other systems aboard the UAV and translates sensor data to the AMPE system.

CHAPTER 6

Handling external forces

6.1 Optimization of path with consideration of wind

Both the planning and control of the UAV should be able to perform when affected by external forces. In this case we take into consideration the wind. Doing this for both the control and planning mean that the MPC as well as the crude planning modules should include some form of wind effect. In order to do so for the MPC, some adjustments inspired by [15] are made to the UAV model constraints. These are also adapted to be included in the MILP formulation. In [15] the components $\mathcal{F}_{w_n} n$, F_{w_e} and F_{w_d} is added to the NED components of the UAV models. The new constraints for the NED differential states of the MPC becomes

$$\begin{aligned}
\dot{N} &= (\cos\theta\cos\psi)u + (\sin phi\sin\theta\cos\psi - \cos\phi\sin psi)v + \\
&\quad (\cos phi\sin\theta\cos\psi + \sin\phi\sin\psi)w + F_{w_n} \\
\dot{E} &= (\cos\theta\sin\psi)u + (\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi)v + \\
&\quad (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)w + F_{w_e} \\
\dot{D} &= (\sin\theta)u + (\sin\phi\cos\theta)v + (\cos\phi\cos\theta)w + F_{w_d}
\end{aligned} \tag{6.1}$$

For the MILP model we bias the velocity constraints toward the direction of the wind. The wind components $\mathbf{F}_w$ are used. The velocity constraints are now modified to

$$\mathbf{v}_{pi}^{T}\,\xi_{kl} \leq V_{pi} \tag{6.2}$$

$$\alpha^{vel}\,\mathbf{v}_{pi}^{T}\,\xi_{kl} + \mathbf{F}_{w} \geq V_{pi} - M_{pkl}^{vel}(1 - b_{pikl}^{vel}) \tag{6.3}$$

$k \in \{1, \ldots, D^{vel}\}, l \in \{1, \ldots, D^{vel}/2\}\ \forall p, q \in P_1^{n_p}, i \in I_0^{N-1}$

$$\sum_{k=1}^{D^{vel}} \sum_{l=1}^{D^{vel}/2} b_{pikl}^{vel} = 1 \tag{6.4}$$

$\forall p, q \in P_1^{n_p}, i \in I_0^{N-1}$

6.2 Implemented utilities

The system is complex as it consists of many modules working in parallel as a looping pipeline providing a singe result. This makes debugging difficult and therefore debugging tools have been created to test each module independently. In addition an UDP service is created to allow the AMPE system to communicate with the separate Simulink simulation. These tools are presented here.

6.2.1 Europa nddl model debug-tool

EUROPA provides a JAVA/C++ interface with a GUI for testing and debugging of models. This tool is expanded upon with helper functions and scripts for fast iteration and debugging of the crude path domain model.

6.2.2 MILP model debug-tool

AMPL allows for the definition of a .run file, which initializes a solver, sets the appropriate options and prints the desired optimization variables upon completion. A simple .run file is created initializing a CPLEX solver.

6.2.3 MPC debug-tool

For this module, a simple c++ application is created. It defines initial conditions in compile time and uses the internal simulation function in ACADO[19] for testing over a given time horizon. Computations are given in the terminal in real time, and simple graphs of all states are provided on completion.

6.2.4 UDP service

The simulations is done locally on one computer, and therefore the common agent, the UAV agents and the external simulator need to communicate throughout some inter-process protocol. To this end an asynchronous UDP library is made using Boost. Boost is a modular set of libraries performing a variety of tasks, and the UDP service uses the libraries asio and thread. Asio, or Asynchronous io, facilitates several protocols and is used to create simple UDP sockets. The UDP peer will create such a socket and start a new thread to listen for incoming UDP datagrams. A list is filled with incoming messages and can be pulled from the UDP peer object asynchronously. To avoid multithreading issues such as race-conditions, the use of this list is protected by a mutex from the thread library of Boost.

Part III

Simulations and results

CHAPTER 7

Simulations and equipment

All simulations in this part will use the same base mission with different parameters and conditions in order to test the different capabilities of the AMPE system. The simulation is done on one computer for all reactors and modules, and therefore any real-time demands can not expected to be met. Instead, steps of 0.25 seconds between each MPC step, using the RK45 method to integrate between steps, while logging computation times and performance. The applicability of the system in a real-world application is then discussed in the next part. Note that any logged computation times will therefore not match up to the simulation times. The linear model presented in chapter 2 is used for the UAV simulations. This is done to thoroughly test the different capabilities the system should have when faced with different situations. Then some work towards using an external simulator with the nonlinear model from 2 is presented. The simulations are performed on a virtual machine Ubuntu, using a Intel(R) Core(tm) i7-4720HQ 3.60 HZ CPU.

7.1 The mission definition

The mission to be performed by the AMPE system is that of the case presented in the introduction. Several marine vessels or sensors called nodes in this thesis, should be visited in order to collect and retrieve data too large for a long range radio link (for example an Iridium satellite link). The mission area, along with the initial node positions, UAV positions and

the base position is shown in figure 7.1. All UAVs should return to the base station when the nodes are visited. The node at north (-600 800 100)^T is moved to (-600 500 100)^T for the last 3 simulations.

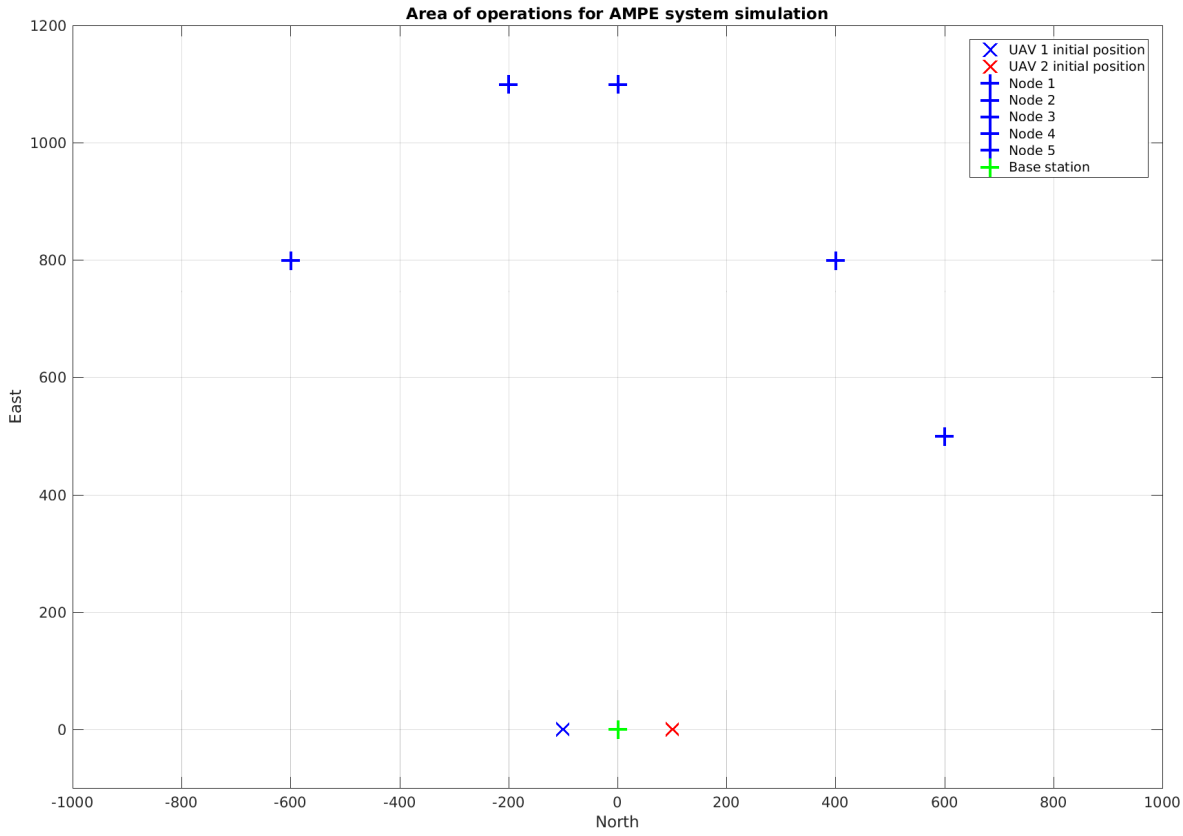


Figure 7.1: The mission area along with initial conditions used for the simulations in this part

7.2 Simulation and system parameters

The MILP connectivity and estimation parameters used, are heavily inspired by [18], and are as follows

Parameter	Value
D^{con}	8
D^{vel}	8
c^{sensor}	2 $Mbitss^{-1}$
C^{max}	$4Mbitss^{-1}$
$t^{separation}$	5 s
$r_1^{acc} r_2^{acc}$	2
D^{vel}	8

As no means of measuring or calculating the actual path loss between UAVs, a simple square function is used for the path loss, and R_{con} is set to 200 m, for a maximum path loss of $4e^4$. The flight area is set to be between -1500 and 1500, -1500 and 1500, and between 50 and -150 in NED coordinates. For the MILP crude path planer, the mission planning horizon is set to 3 minutes and 30 seconds with a sampling time of 5 seconds which N 42 time steps. V_{min} is set to 18m/s, and V_{max} is set to 19 m/s. This is quite strict, but keeps the speed of the UAVs close tho the trim conditions around which the system is linearized so that the model stays accurate. The distance d_x when a waypoint is met is set to 10, and anti-collision distances d_N, d_E and d_D is set to 50 m.

CHAPTER 8

Simulating static missions

8.1 Simulating a static mission using EUROPA

The mission was first simulated using EUROPA as the crude path planer module. The model presented in section 5.1.2 is used, and the resulted plan is grounded at early start times and presented in figure 8.1.

The plan is created efficiently. The module uses 1.85 seconds to deliberate the plan. However the plan do not seem to be optimal. Instead of trying to handle several close nodes, UAV 2 only handles one node. For this and all further simulations, the MPC is used as both a refined path planer, and a controller.

The trajectory is clearly not time optimal, the left UAV (UAV 1) returns after two minutes and 40 seconds. The simulation were stopped at the final approach. Some other system would take control at this point. To asses the MPC performance, the rest of the flight states are given in the next few pages.

8.1.1 Expanding to dynamic planning

The results of dynamic mission simulations using the EUROPA reactor would be very interesting as the EUROPA reactor is many times more computationally efficient than the MILP

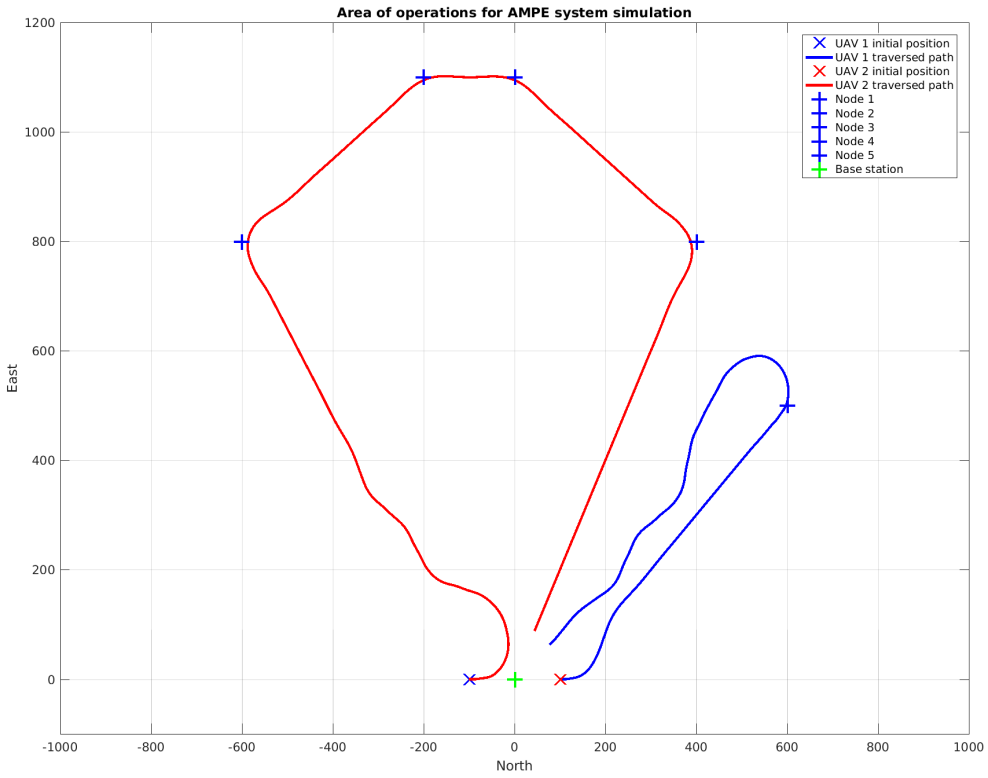


Figure 8.1: The resulting traveled trajectory of the EUROPA crude path test.

reactor, and therefore more suited in the dynamic case than the static. Sadly, because of technical issues with local compilation of some EUROPA code, this simulation could not be performed. The issue is related to the open issue #172 on Github [2].

8.2 Simulating a static mission using MILP and data flow constraints

The system will be tested using the MILP crude reactor, and otherwise, the mission is similar to the previous. The left UAV (UAV 1) is assigned a buffer size of two complete messages, and all constraints from section 5.2 is included. The MILP computed an optimal path before the mission started which is given in figure 8.9. The mission is then started and the UAV follows the path using the MPC for combined refined planning and control. The resulting traveled path is shown in figure 8.10.

From figure 8.10, it is seen that the MPC are able to follow the path given by the MILP crude path module in a smooth and accurate manner. The figures 8.11 and 8.12 shows how the

path is accomplished by the UAVs without excessive actuation, and that the inputs changes smoothly. The system aimed to minimize actuation wear and tear, and battery usage, and the figures shows good results towards this end. The red dotted lines shows the trim condition, and the black lines are the constrains on the control.

8.2.1 Expanding to dynamic planing

The plan of this simulation was only deliberated initially and not maintained online. This is because, for this simulation with the data flow constraints, solving the MILP problem could take anywhere from 2 hours to more than a day based on the mission configurations. Therefore, for the rest of the simulations, the data flow constraints are not included and the buffer size is assumed to be sufficient for ferrying all data.

8.3 Re-planing capabilities in a static mission

In this section a brief simulation will be presented in which the MPC is tuned to perform poorly. Figure 8.13 shows the plan after the first waypoint, which is the same as the plan dispatched at time 0. It can be seen inn this plot that UAV 2 overshoots the turn and ends up behind in the plan. This is picked up on the next crude path plan, and as seen in figure 8.14. UAV 1 is now set to handle both the upper nodes, and UAV 2 returns to base. Figure 8.15 shows the completion of this mission. As the MPC is purposely tuned poorly, no flight states are included in these results.

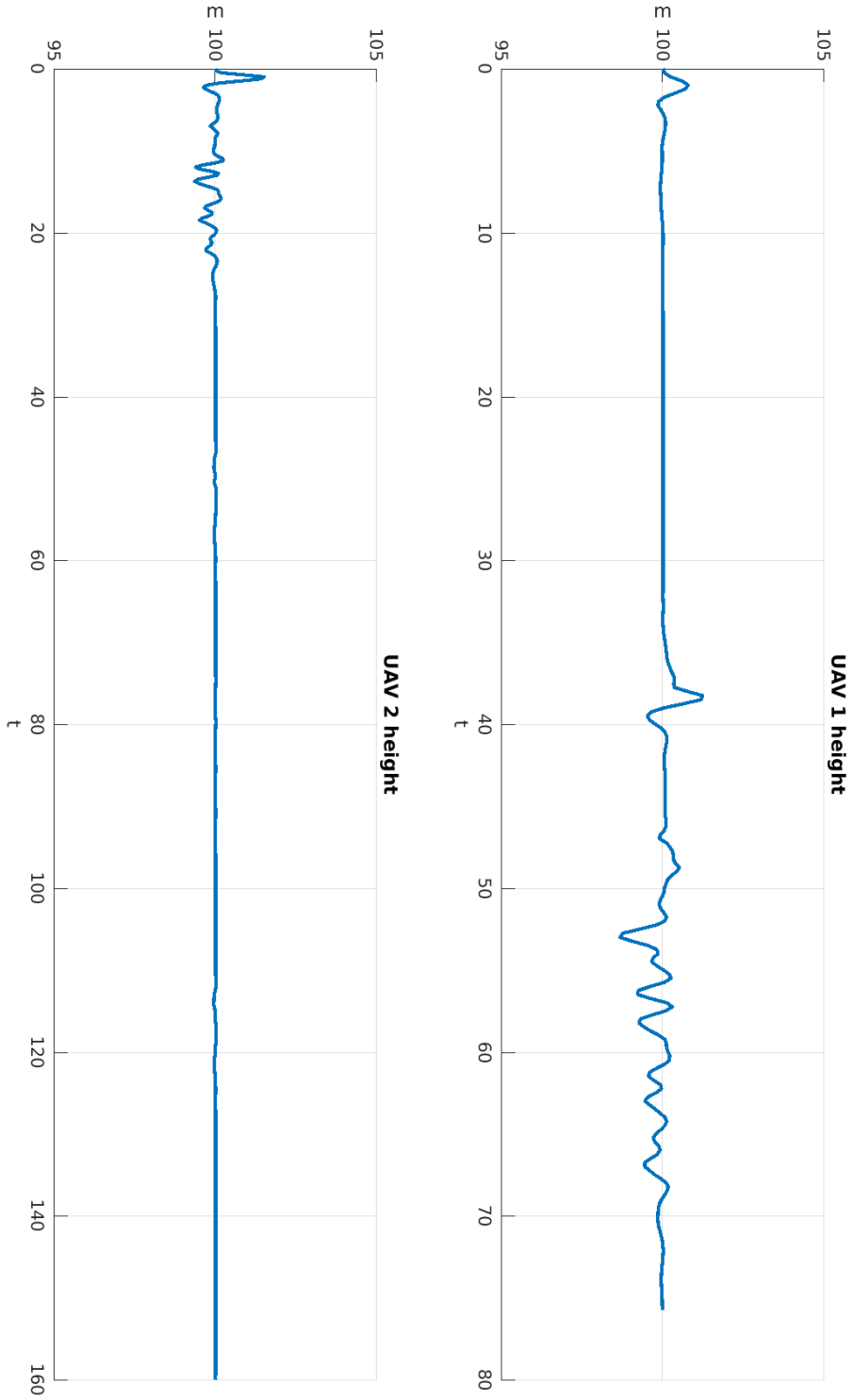


Figure 8.2: The EUROPA path simulation. The dotted red line represents the trim condition, and the black lines the MPC constraints.

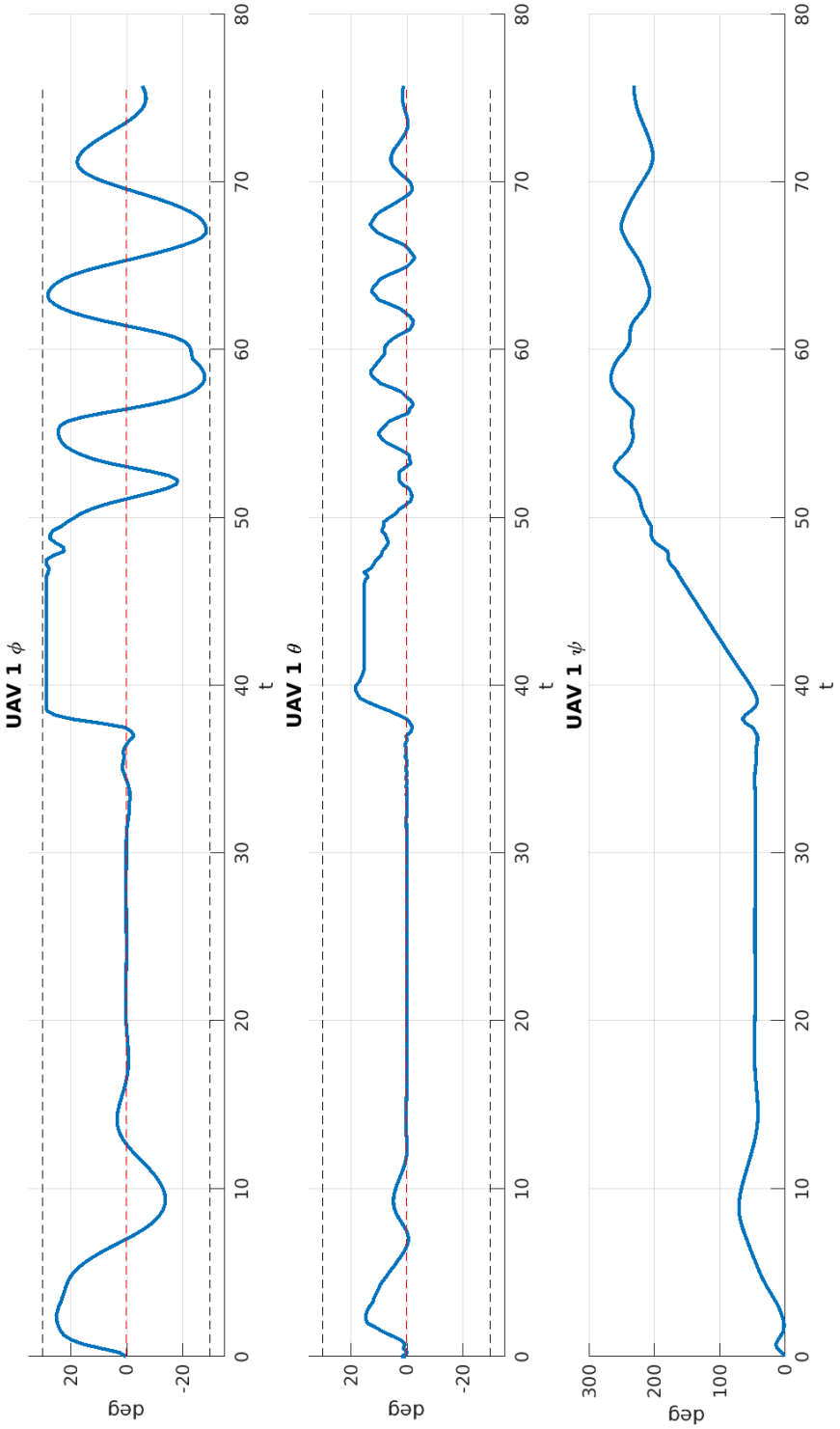


Figure 8.3: The EUROPA path simulation. The dotted red line represents the trim condition, and the black lines the MPC constraints.

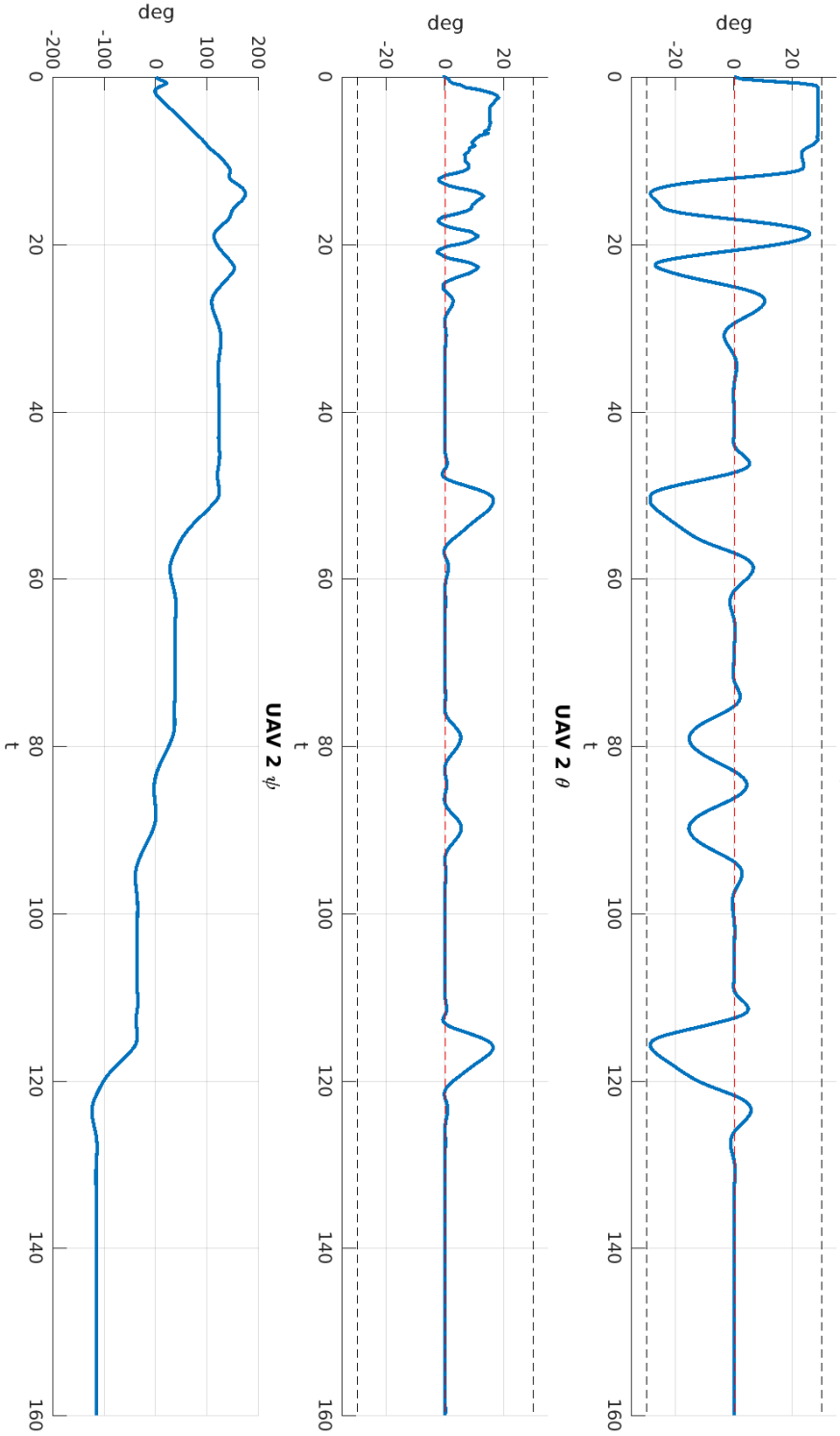


Figure 8.4: The EUROPA path simulation. The dotted red line represents the trim condition, and the black lines the MPC constraints.

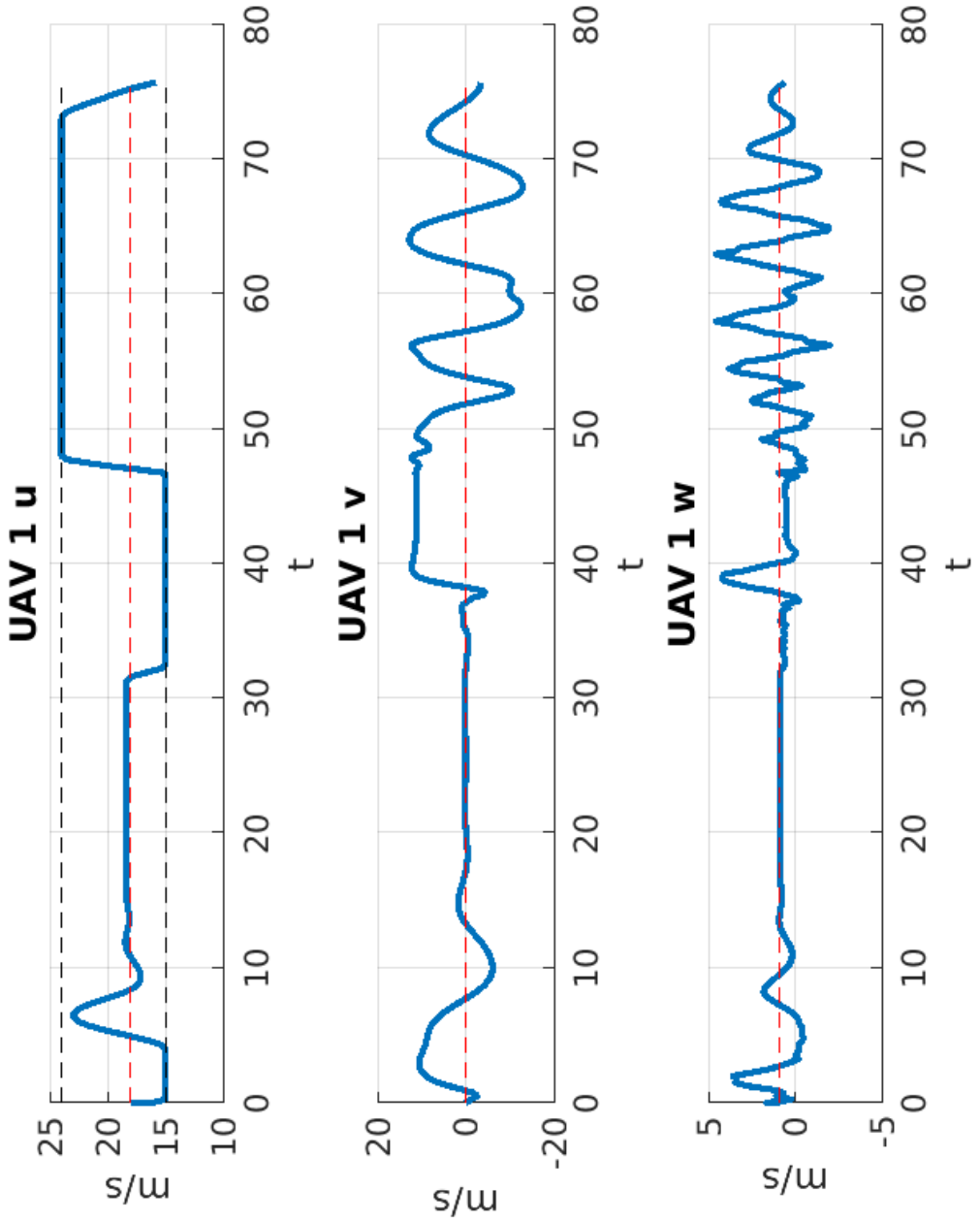


Figure 8.5: The EUROPA path simulation. The dotted red line represents the trim condition, and the black lines the MPC constraints.

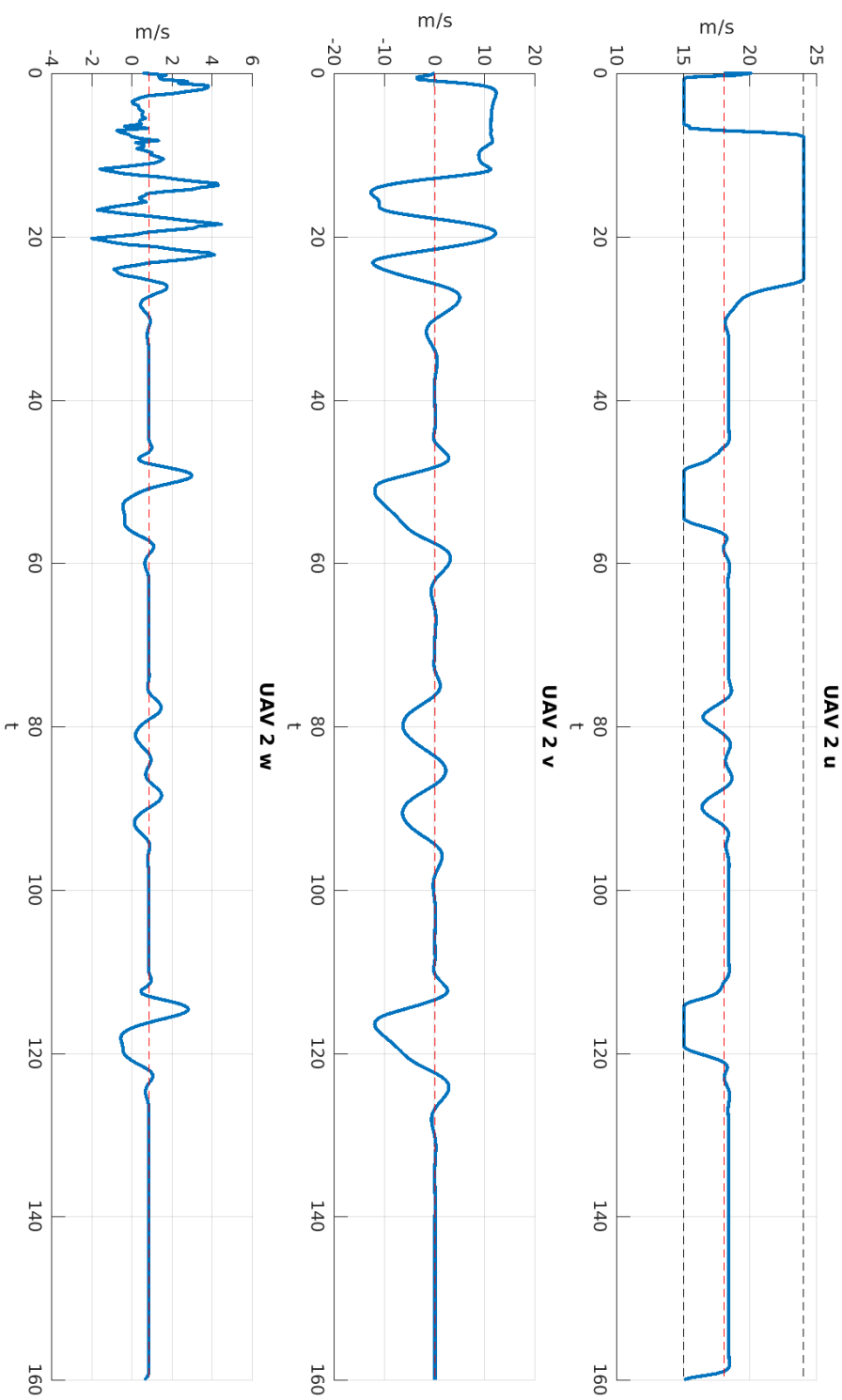


Figure 8.6: The EUROPA path simulation. The dotted red line represents the trim condition, and the black lines the MPC constraints.

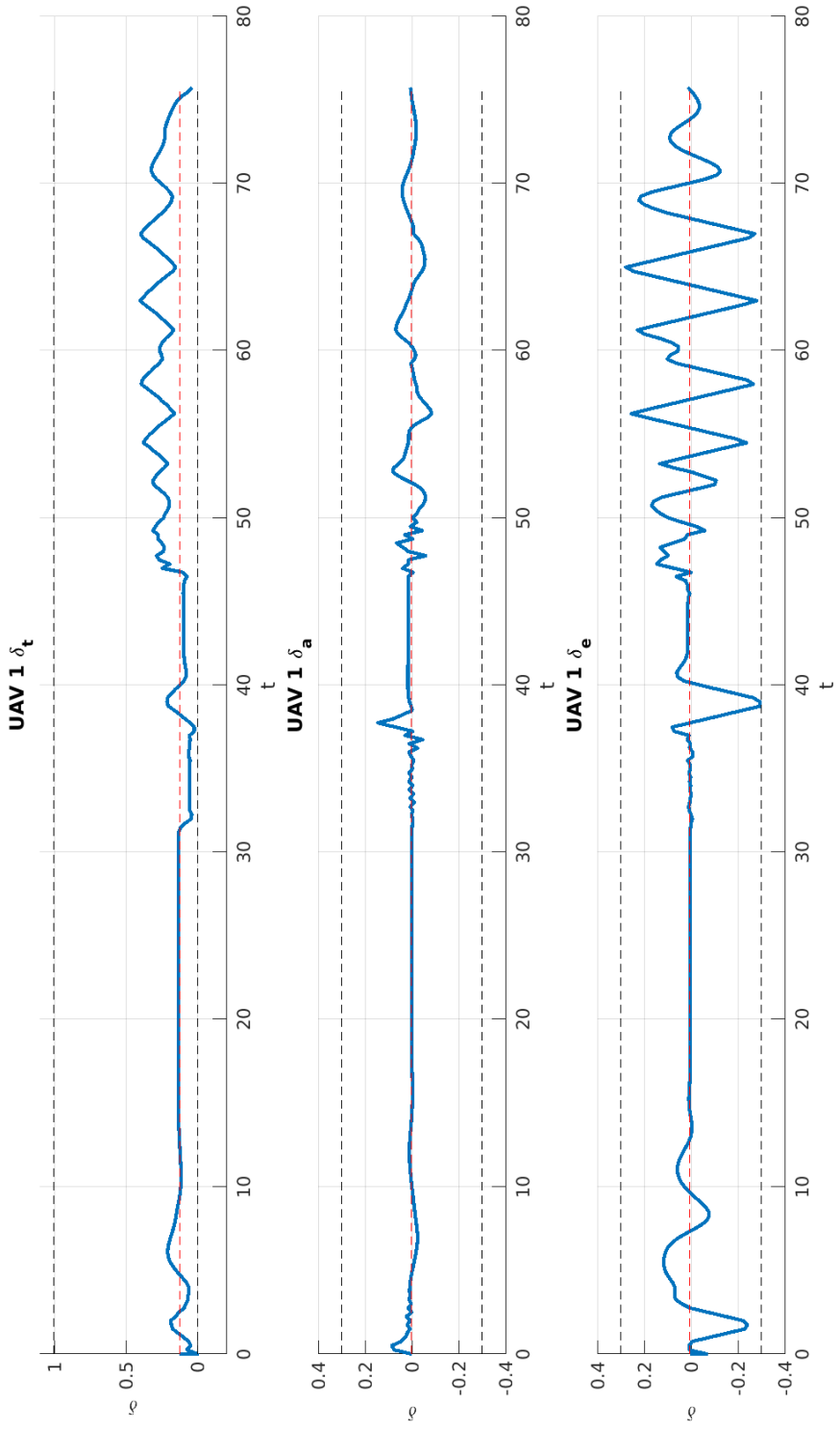


Figure 8.7: The EUROPA path simulation. The dotted red line represents the trim condition, and the black lines the MPC constraints.e

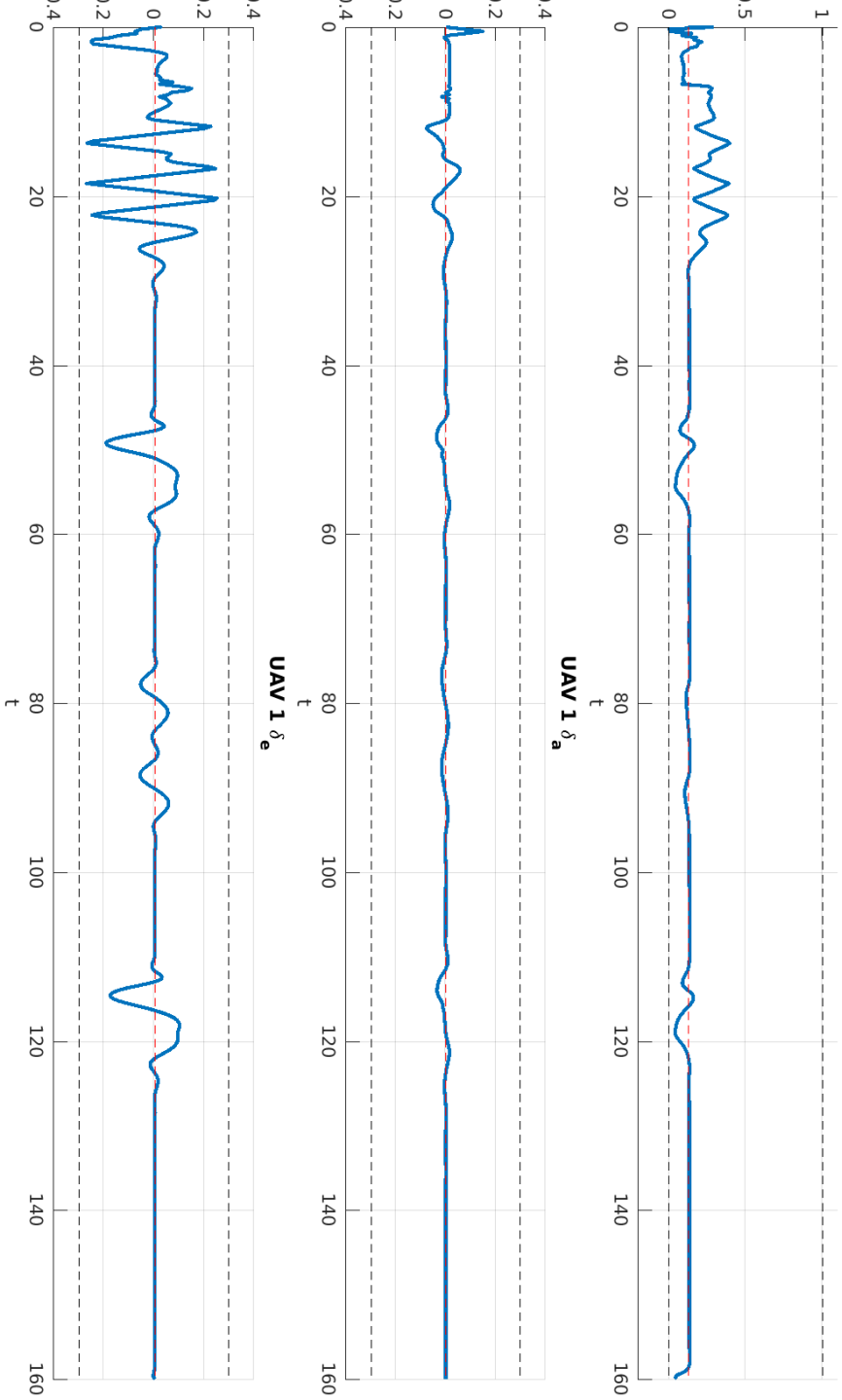


Figure 8.8: The EUROPA path simulation. The dotted red line represents the trim condition, and the black lines the MPC constraints.

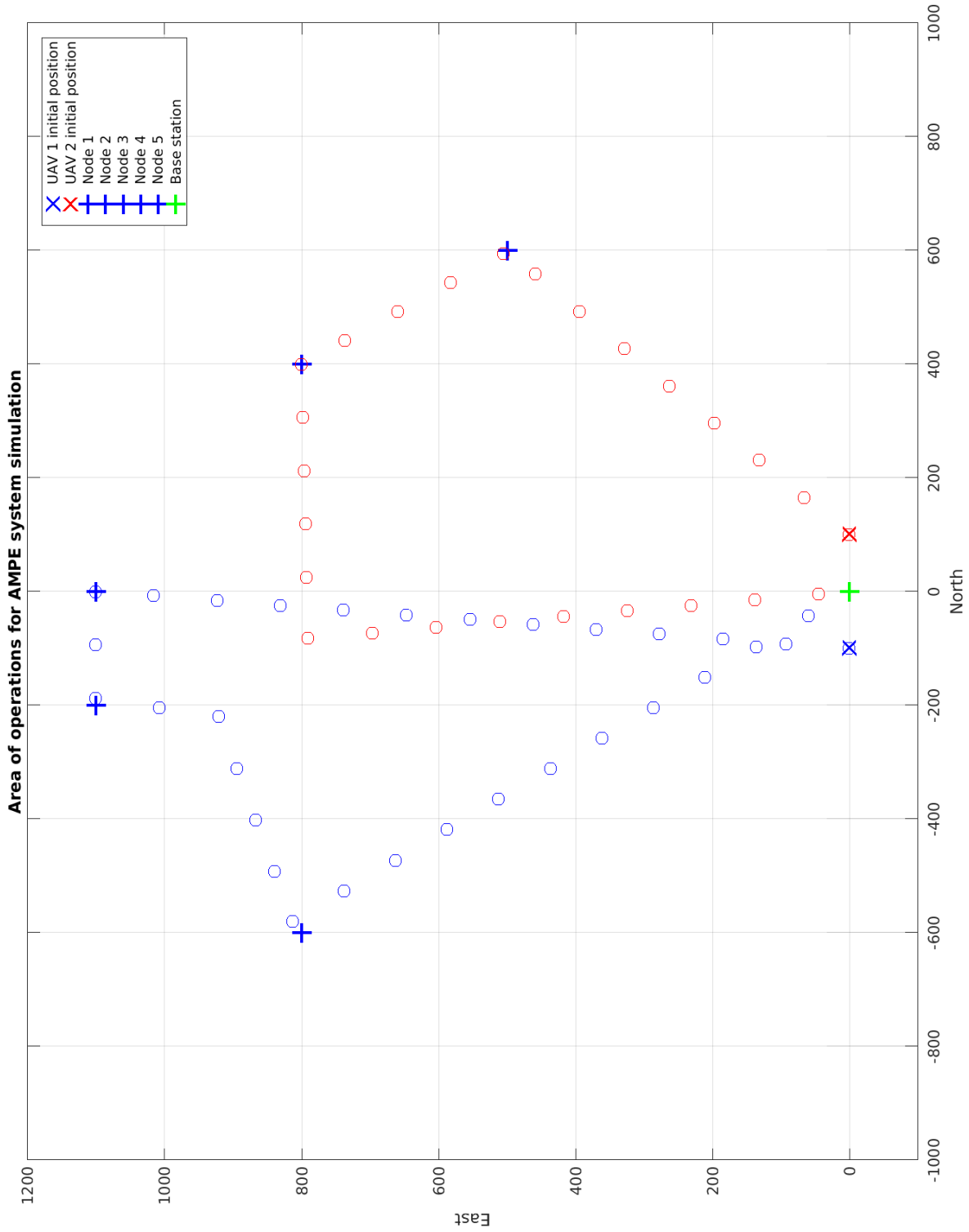


Figure 8.9: The initial crude paths calculated by the MILP module using data flow constraints.

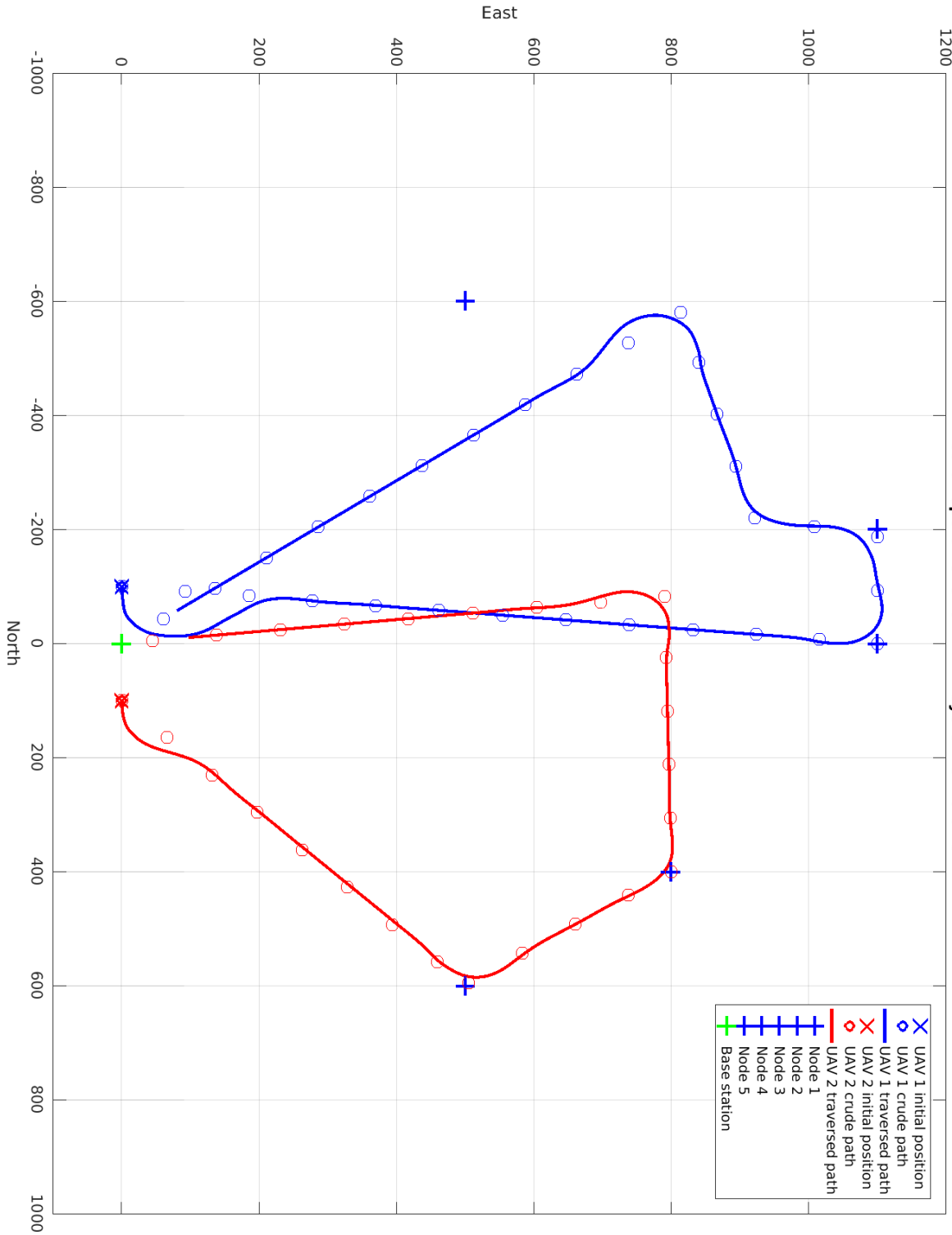


Figure 8.10: The traveled trajectories following the initial crude paths calculated by the MILP module using data flow constraints.

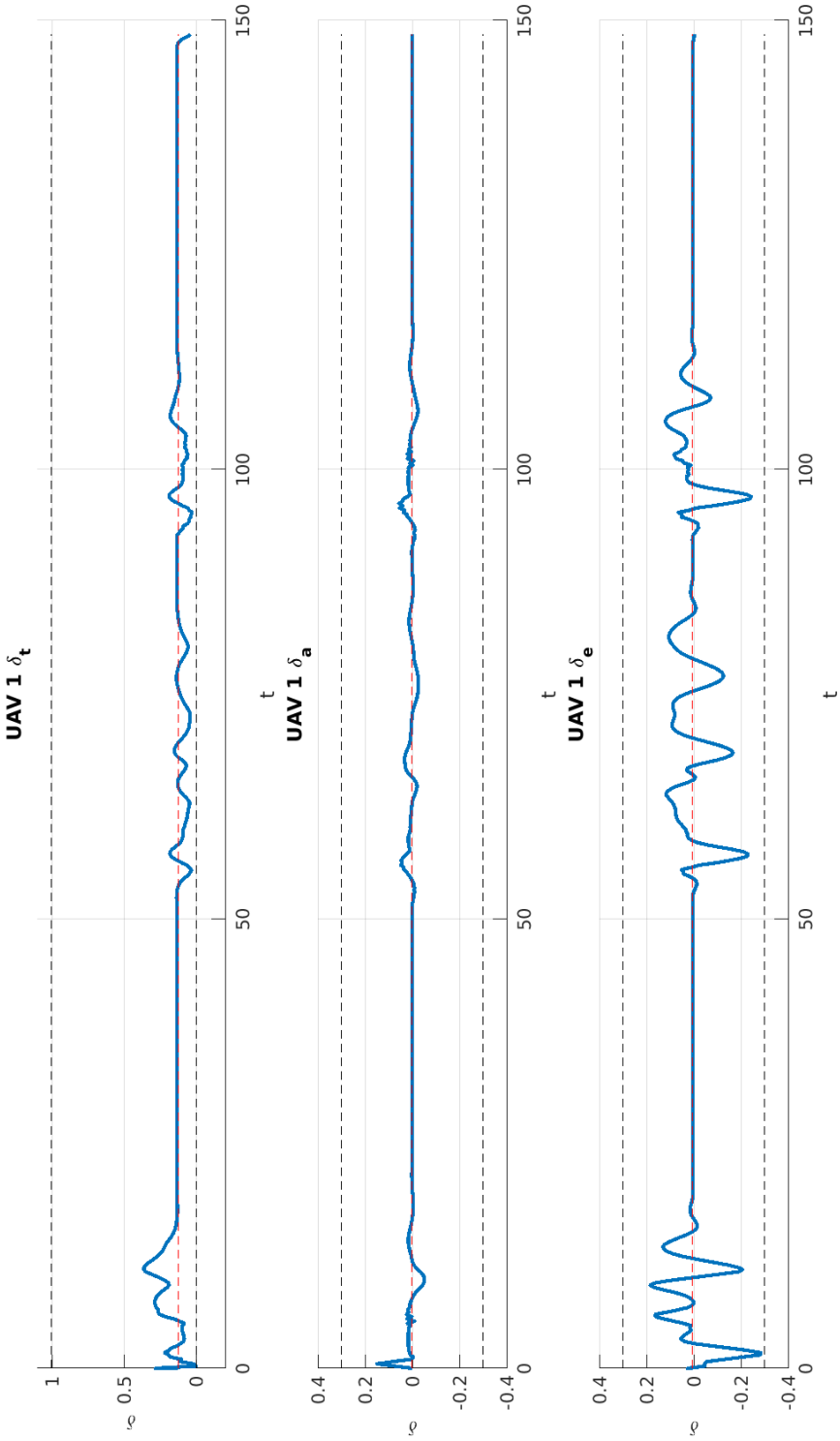


Figure 8.11: The controls of the UAV 1 following the initial crude paths calculated by the MILP module using data flow constraints. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

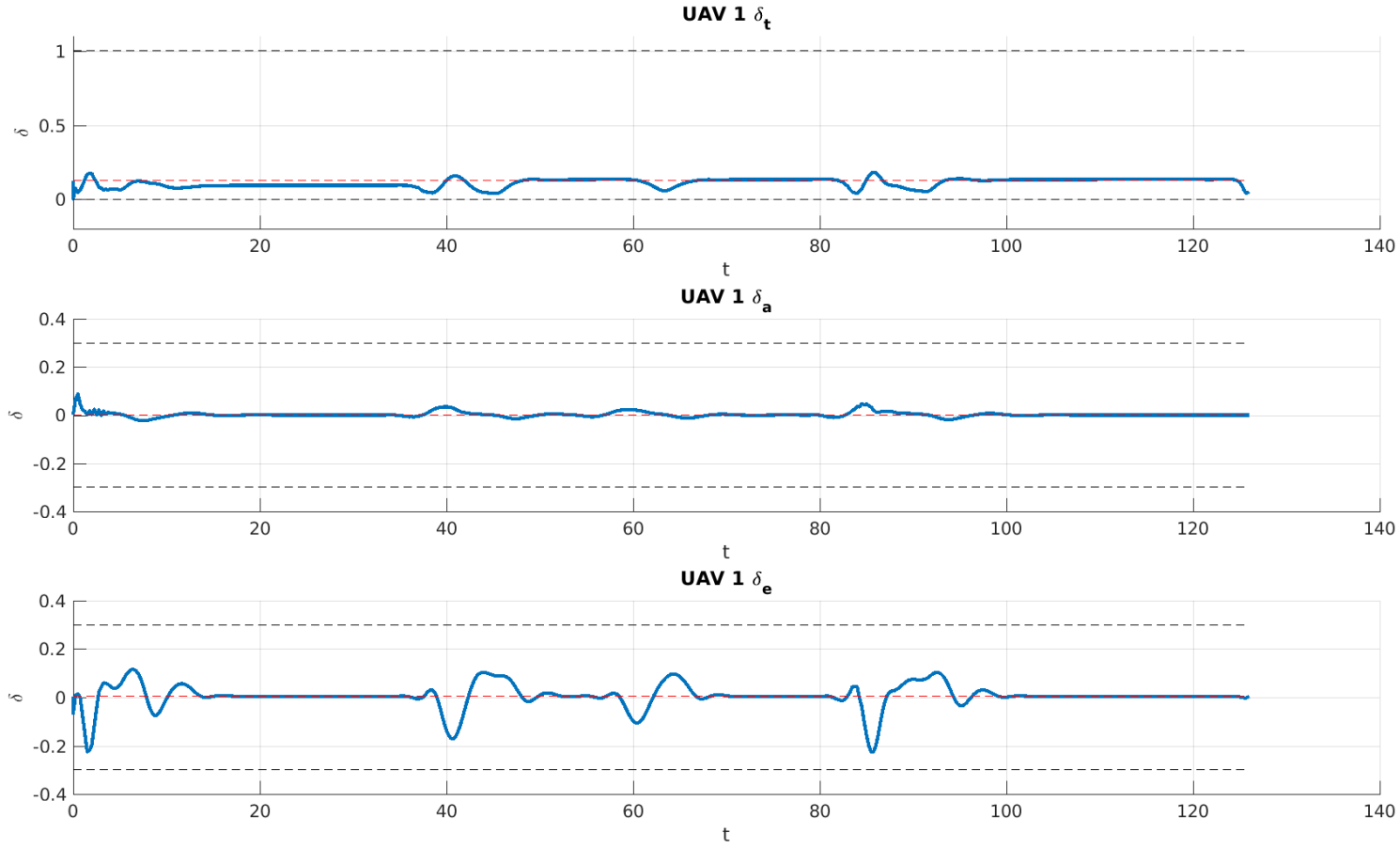


Figure 8.12: The controls of the UAV 2 following the initial crude paths calculated by the MILP module using data flow constraints. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

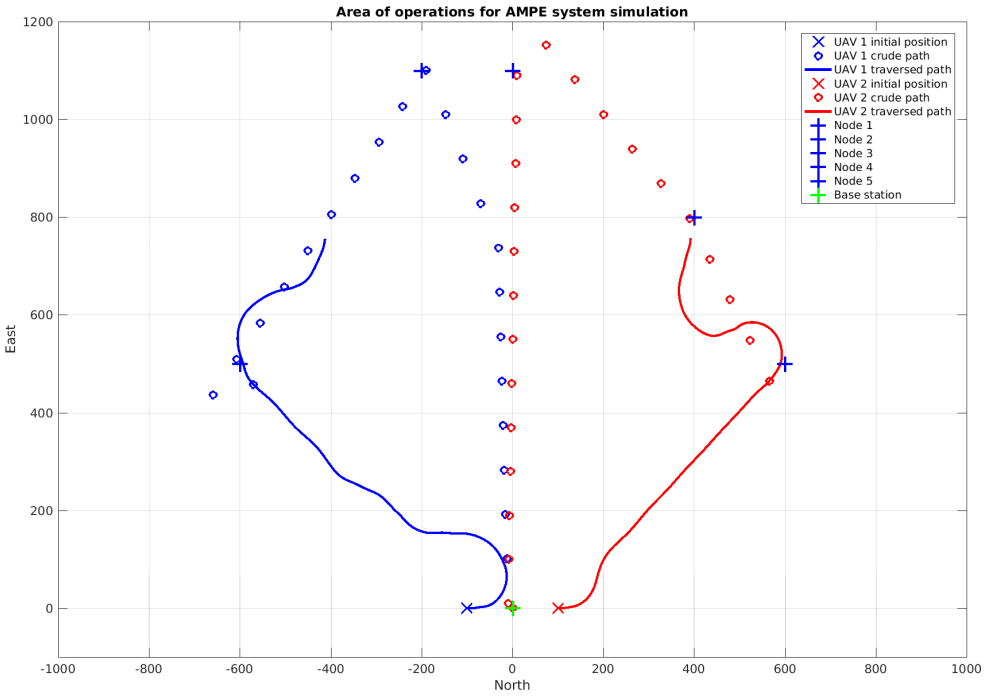


Figure 8.13: The static mission with a poorly tuned MPC. At this point the plan is unchanged from the initial.

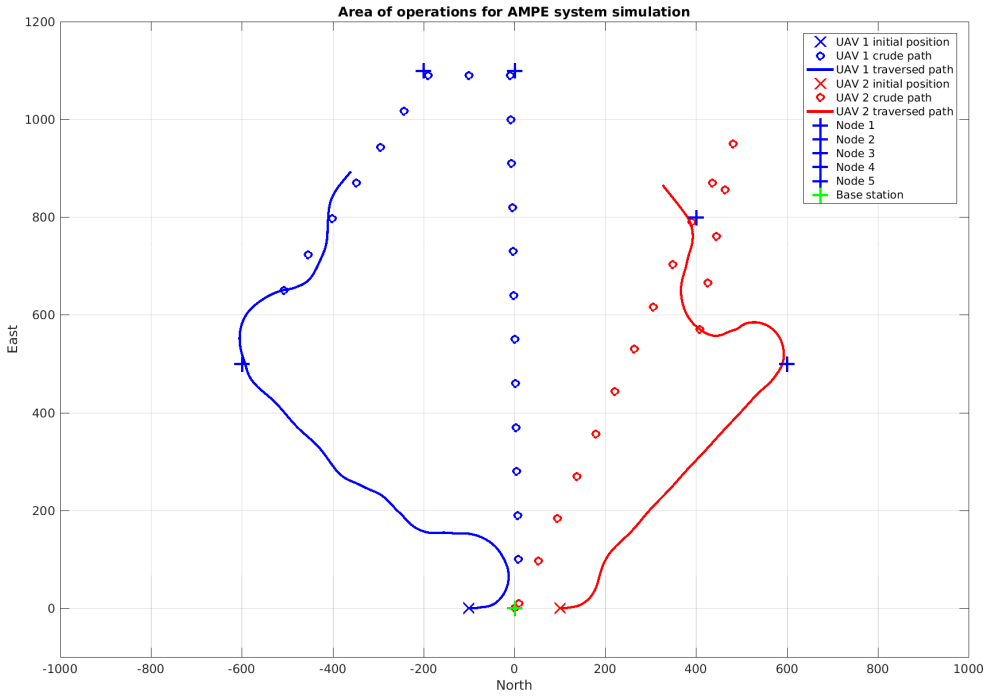


Figure 8.14: The static mission with a poorly tuned MPC. The plan has changed to accommodate the delay of UAV 2.

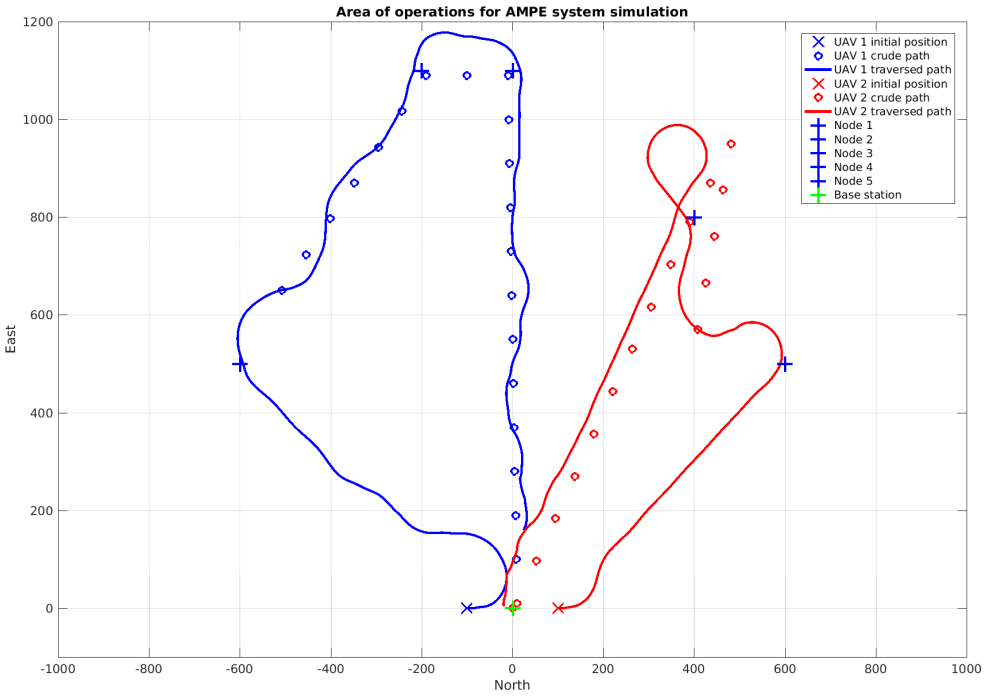


Figure 8.15: The static mission with a poorly tuned MPC. The plan has changed to accommodate the delay of UAV 2.

CHAPTER 9

Simulating dynamic missions

In this chapter the simulations will include some dynamic elements to the missions. First the battery consumption of one UAV will be increased throughout the mission, and then a mission configuration is used in which some of the nodes are moving at a speed of 8 knots.

9.1 Simulating a mission with dynamic resources consumption

In order to test how the system adapts to changing resources, the battery is set to drain increasingly faster than the crude path planner assumes it will. UAV 1 will not be able to complete the same mission as planed in the previous simulations. Figure 9.1 shows the start of the simulation with the initial plan. The planer is started again as the mission starts, and deliver the same plan. However, when the crude path module start deliberation on 40 seconds, it delivers the plan about 46 seconds into the mission, and the resources are too low. The plan changes, and UAV 2 is now set to handle all nodes. The plan at 35 seconds is illustrated in figure 9.2

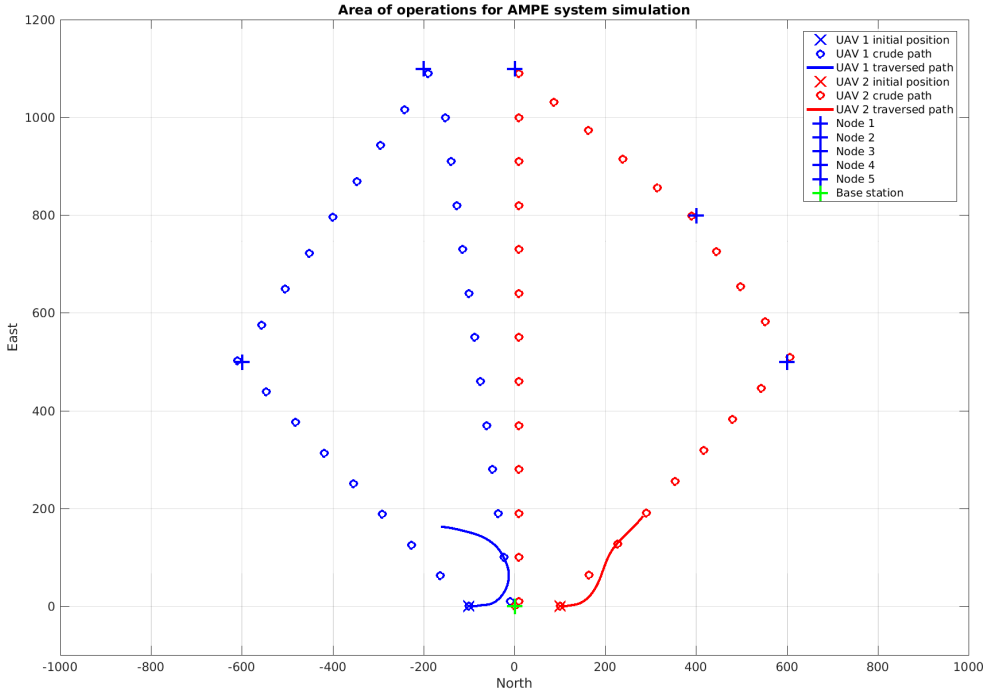


Figure 9.1: The start of the constrained resources simulation. The battery consumption of UAV 1 increases with time and the plan should change during the mission.

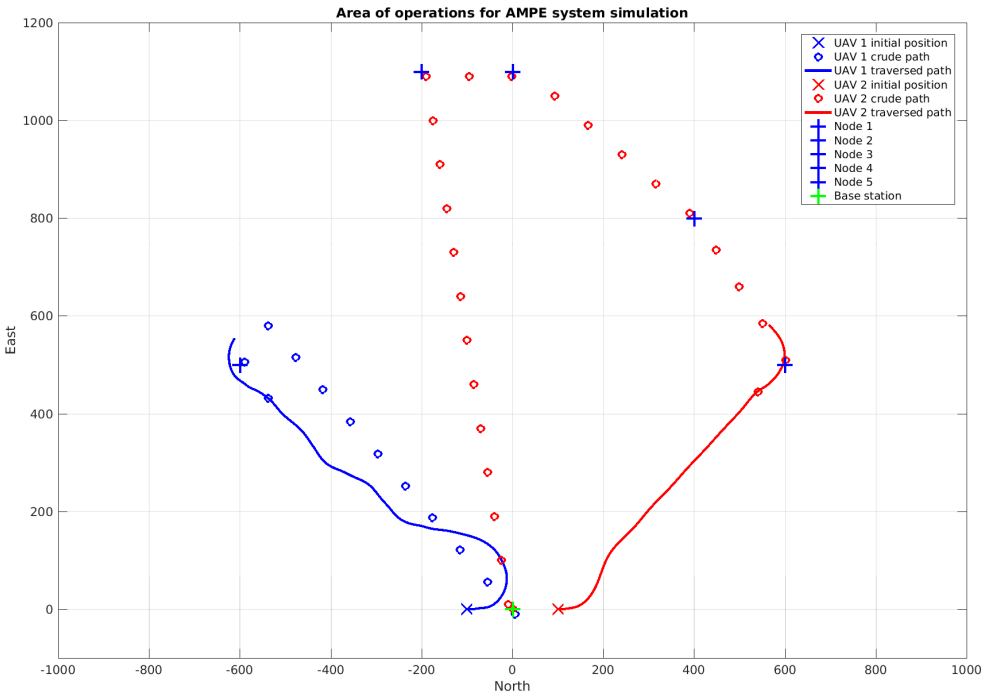


Figure 9.2: 46 seconds into the constrained resources simulation. The battery of UAV 1 is drained at triple speed and the plan is changing during the mission.

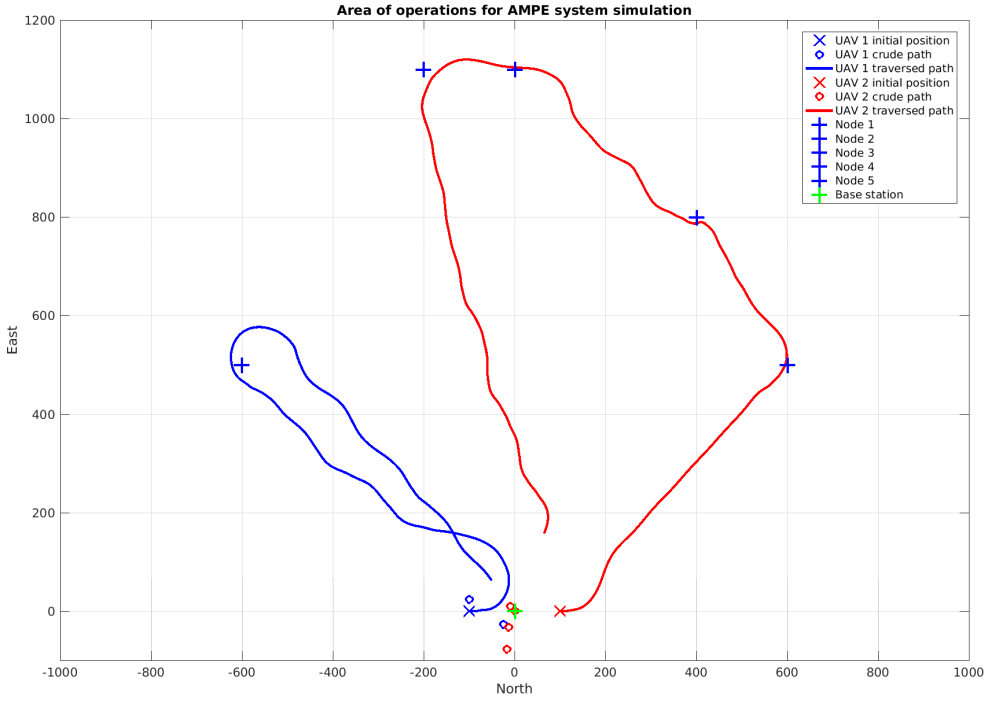


Figure 9.3: The end of the constrained resources simulation. The battery of UAV 1 is drained at triple speed and the plan has changed during the mission.

After about 1 minute and 45 seconds seconds the mission is completed as the UAVs approach the base station. This stage is shown in figure 9.3. The computation time of the individual MPCs and the MILP crude path planner is shown in figure 9.4. For the crude path planer reactor, it is interesting to notice that after the initial long calculation times, the times drop significantly when UAV 1 reaches the first node and is deemed to low on battery to handle more nodes. This illustrates how the MILP solver depends on the complexity of the mission mode, given the current state. The plot also shows how the initial deliberations exceed 300 seconds, which poses a problem to the applicability on a online system. The MPC calculations stay somewhat stable excepts some serious increases. These happens at plan changes and sharp turns when the objective function becomes larger.

9.2 Simulating a mission with dynamic node positions

One goal of the system is to investigate the possibility of replacing the AI modules of T-REX with mathematical optimization algorithms, and still use the system for dynamic missions with moving nodes. This simulation illustrates the system performance when the two top nodes are given a velocity of 8 knots (about 4.5 m/s). The top right node will move 180 degrees directly south, while the top left node will move at 45 degrees north east. Again the initial mission plan is the same, but after 60 seconds the plan changes several times. Figure 9.5 shows the plan which are synchronized from the crude path planer at 60 seconds.

Notice that the deliberation time have been extensive, which causes a big lag from the state dispatched to the crude path reactor to the state when the plan is dispatched. This provides a challenge to the MPC which will in turn get a significantly higher objective function. The controls of UAV 1 illustrated in figure 9.9. At 60 seconds, when the plan is dispatched there is a large increase in rapid control moves, which are undesirable when seeking to limit wear and tear as well as battery consumption. However, the plan is able to adapt to the moving nodes, and after 75 seconds a new plan is dispatched shown in figure 9.6. The new plan uses the positions at 60 seconds, and UAV is now given responsibility of the upper nodes, while UAV 2 takes care of the left node which UAV 1 have missed due to the MPC turning early in the bend at that node.

Then again after another 20 seconds a new plan is dispatched. At this point UAV 1 have taken a right hand turn, and is closer to the left node. The new plan gives the node to UAV 1 and figure 9.7 shows the UAVs after following this plan for some 10 seconds. The last stage of the plan is shown in figure 9.8, where the plan is simply for all UAVs to return to the base station. At this point some other system takes control and handles the approach and landing at the base station.

For this mission, all flight states will be illustrates the figures 9.10 to 9.16. These are included to show some interesting differences from the EUROPA planner simulation. The change of paths causes a much more sporadic behaviour and it can be seen especially as the first new plan is dispatched at 60 seconds that the increasingly rapid actuation is represented in all states. This is normal in the case of the 60 second plan change as the new plan is drastically different. But this phenomena is present also in the last 20 seconds when the UAVs does their approach, but before they have the base station in the time horizon of the MPCs. This seems to be caused by a timing challenge caused by the design of objective function for the MPCs. In figure 9.17 the computation times of the MPCs and MILP module are shown. The MILP times are especially high during the first 10 iterations of the plan, almost averaging on

500 seconds.

The mission is successful in this simulations with two moving nodes, but with more dynamics in the nodes position, the slow deliberations of crude paths can cause serious issues. The delay when between node position sensing and the plan being dispatched can cause the UAV to miss the node entirely. The node will not register as visited and the planner will attempt to visit it again.

Figure 9.18 shows a mission which illustrates this point. In this mission, all nodes except the upper right is moving. UAV 1 misses the first node on the left, and continues towards the upper nodes. The plan changes at the top like in the successful mission, but this time the node at the bottom remains. This worsen when UAV two misses the upper, now right, node. The result are that both nodes follows the path past a node, misses it, flies towards the next nodes before doubling back and missing the node again.

9.3 External simulator

Work was done to implement the AMPE system as a planning and control system in an external simulator created by Kristoffer Gyrte in [16]. Sadly I was not able to develop this into a stable system in terms of reliable functionality during the appointed time of this thesis, and it is therefore left out of this chapter. However, experience using the simulator shows that the system is also not stable in terms of control when simulating in real time using the MPC as a controller. This is because of the long calculation times. As shown in the above results, the MPC can sometimes use as much as 5 seconds to calculate one control input. This is unsuitable for the rapid flight dynamics of the UAV, which quickly starts to oscillate increasingly about the given path and looses altitude. Solutions, and other possible reasons of instability is discussed further in chapter 10. Another setup was attempted using the MPC as a pure refined planning module, and use an external PID controller provided in the simulator for control. Even though no logged data is available for these tests, experiences with this setup will be included here as it facilitates further work, and affects the conclusion of this thesis.

The step length of the MPC was increased to 2 seconds as this is within some margin from the average calculation times. A Runge-Kutta45 integrator is then used with the input to find the intended waypoint of the first control provided by the MPC. This is done again with the next two inputs to keep as a buffer in case the next calculation exceeds the 2 seconds until the UAV should pass the first waypoint. The combined time of these operations becomes the deliberation time of the reactor, and tended to average on 1.8 to 2 seconds. Therefore the first

waypoint in the buffer was often used but seldom the second. The height of the waypoints are extracted and used as a height offset, and then line-of-sight guidance as described in [14] is used to set a heading course reference. This setup provided stable flight and was able to meet waypoints during the short simulation tests that were achieved. Wind forces were also applied and terms to account for this discussed in section 6.1 were set to match the wind speed and direction. As the NED position relative to the current path waypoint is part of the MPC objective function, the controller becomes more and more erratic in terms of control amplitude when the distance is too far. This makes the wind terms in both the crude path planner, and the MPC essential. The flight path tended to be smoother when expected air and ground speed was not the same.

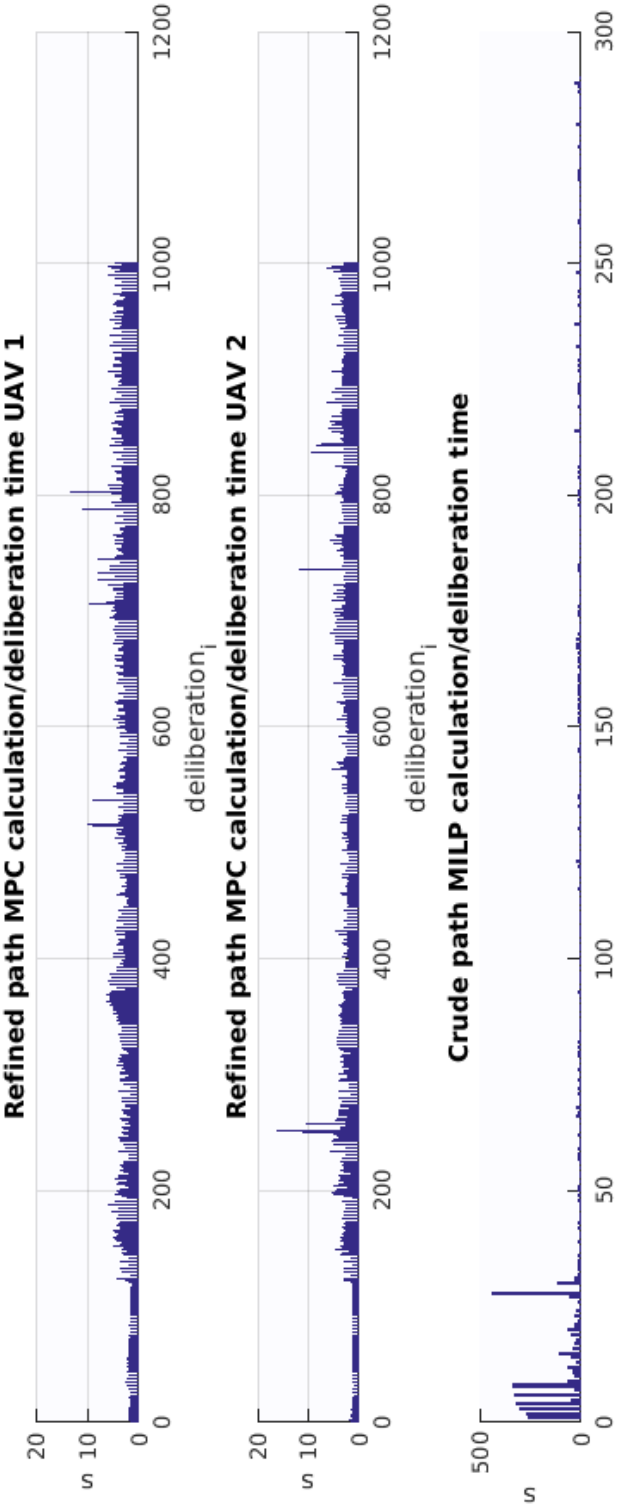


Figure 9.4: The computation time of the MPCs and MILP reactor during the constrained resources simulation.

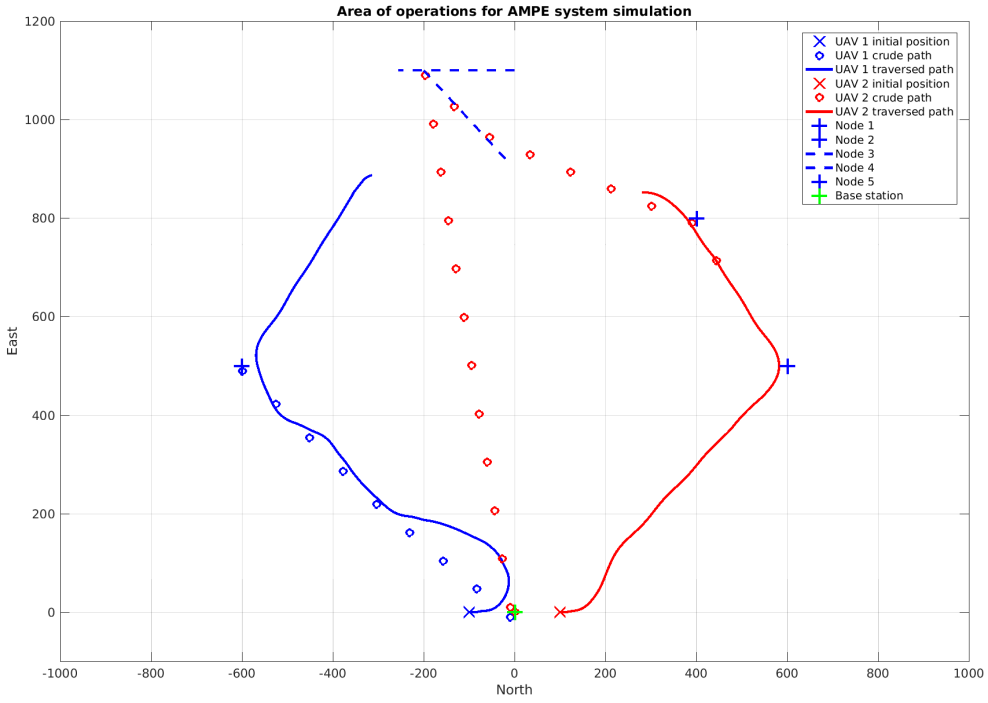


Figure 9.5: This figure illustrates the moving nodes simulation. after about 60 seconds the first new plan is deliberated by the MILP module telling UAV 1 top return.

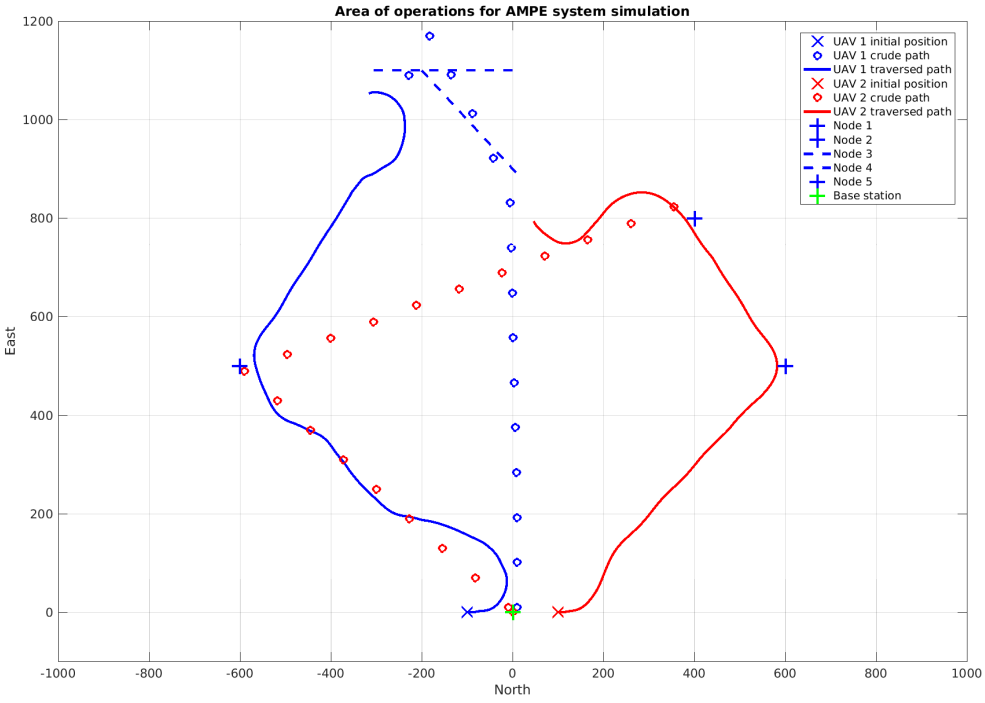


Figure 9.6: This figure illustrates the moving nodes simulation. After about 60 seconds the first new plan is deliberated by the MILP module telling UAV 1 top return.

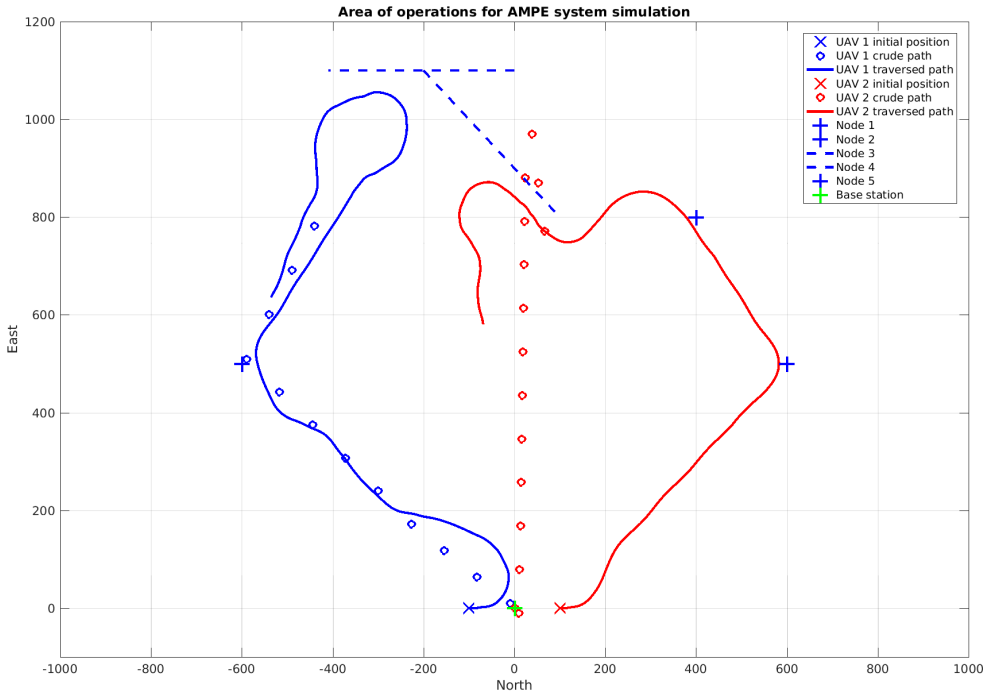


Figure 9.7: This figure illustrates the moving nodes simulation. After about 75 seconds a new plan is dispatches, using the positions at 60 seconds.

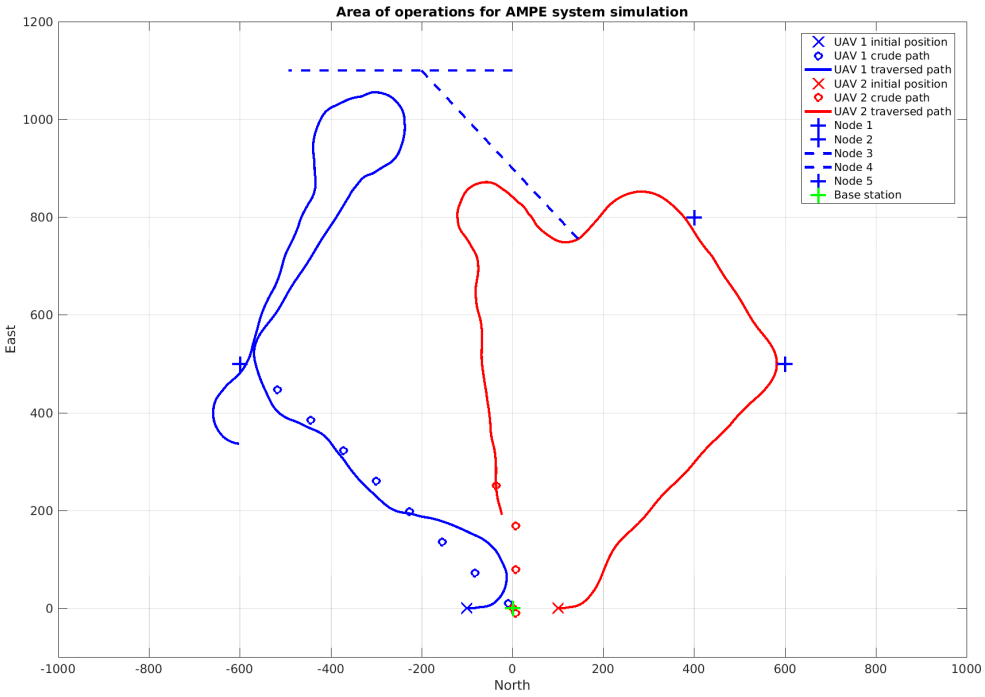


Figure 9.8: This figure illustrates the moving nodes simulation. At this point of the simulation the mission some landing system takes control.

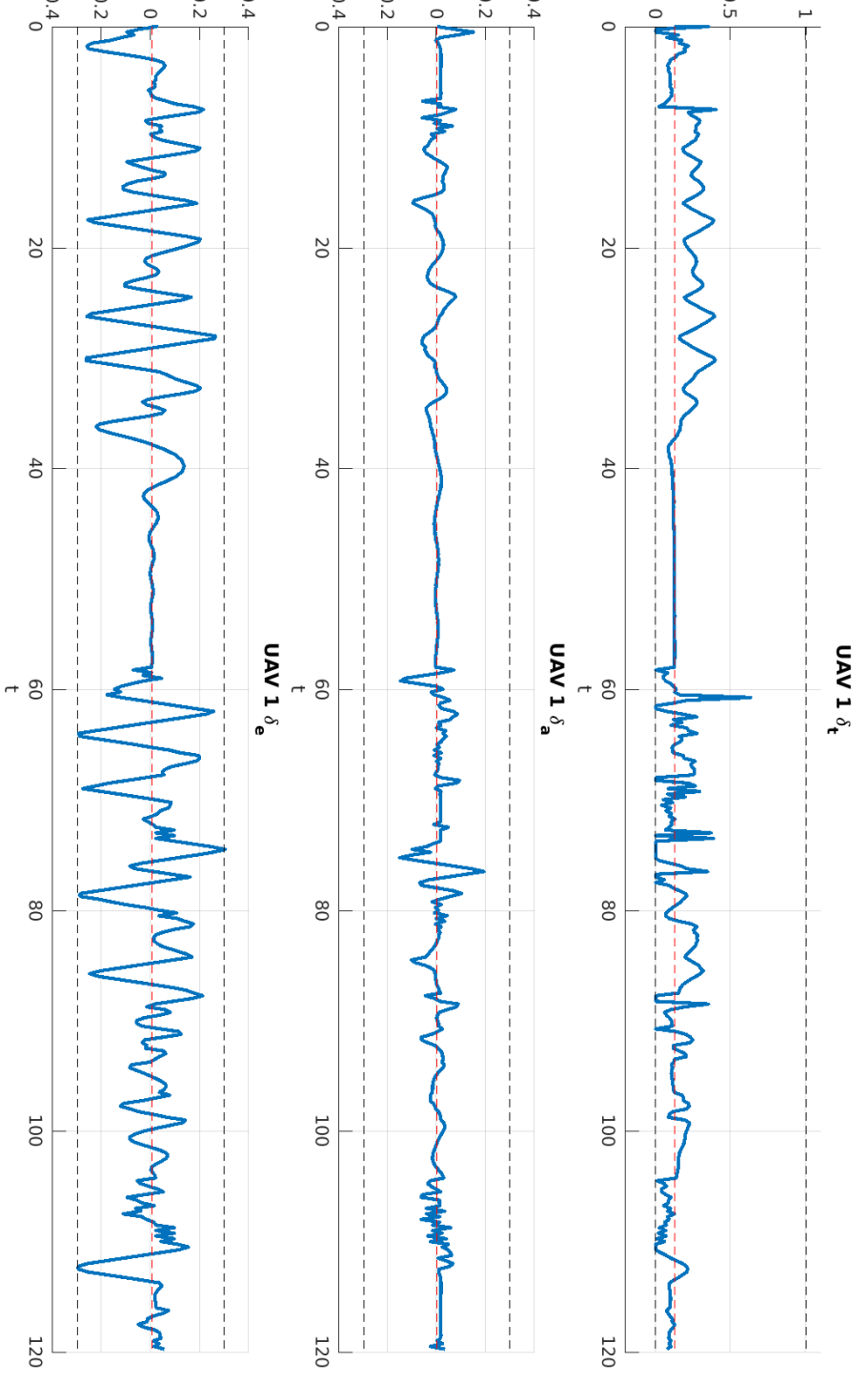


Figure 9.9: The controls of UAV 1 during the moving nodes mission. Notice the increase in rapid control moves when a new plan is received at 75 seconds.

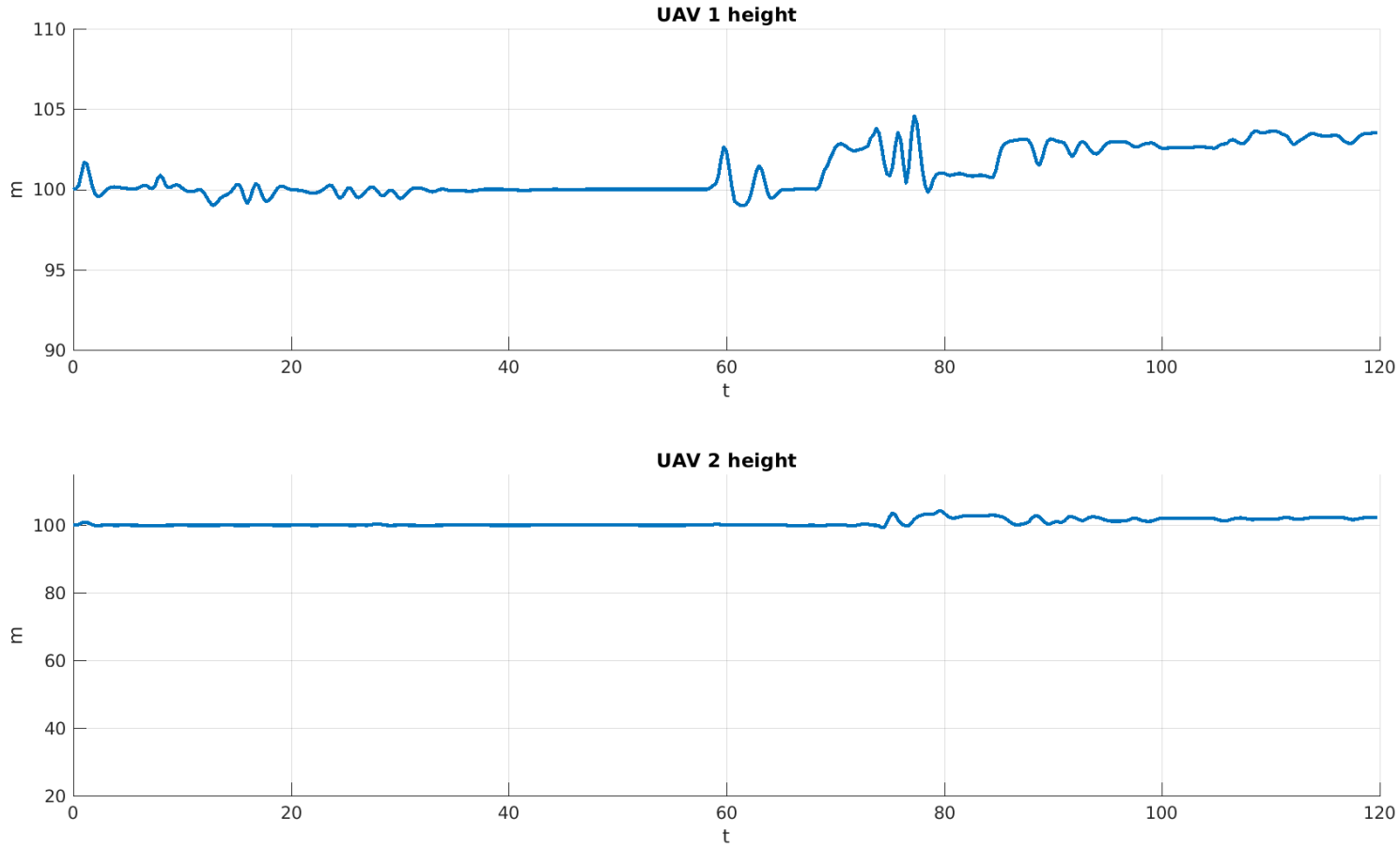


Figure 9.10: Data from the dynamic node position simulation. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

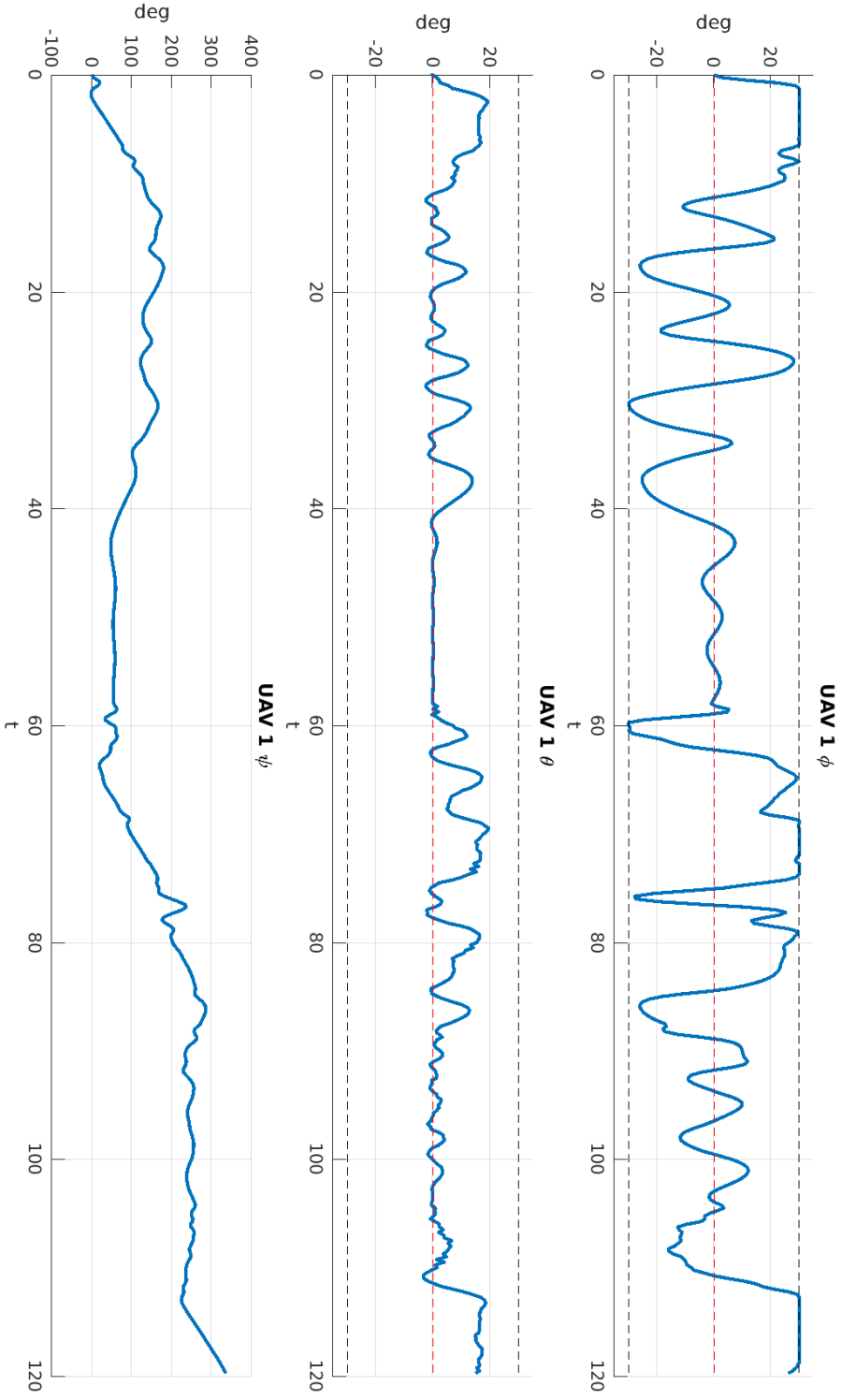


Figure 9.11: Data from the dynamic node position simulation. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

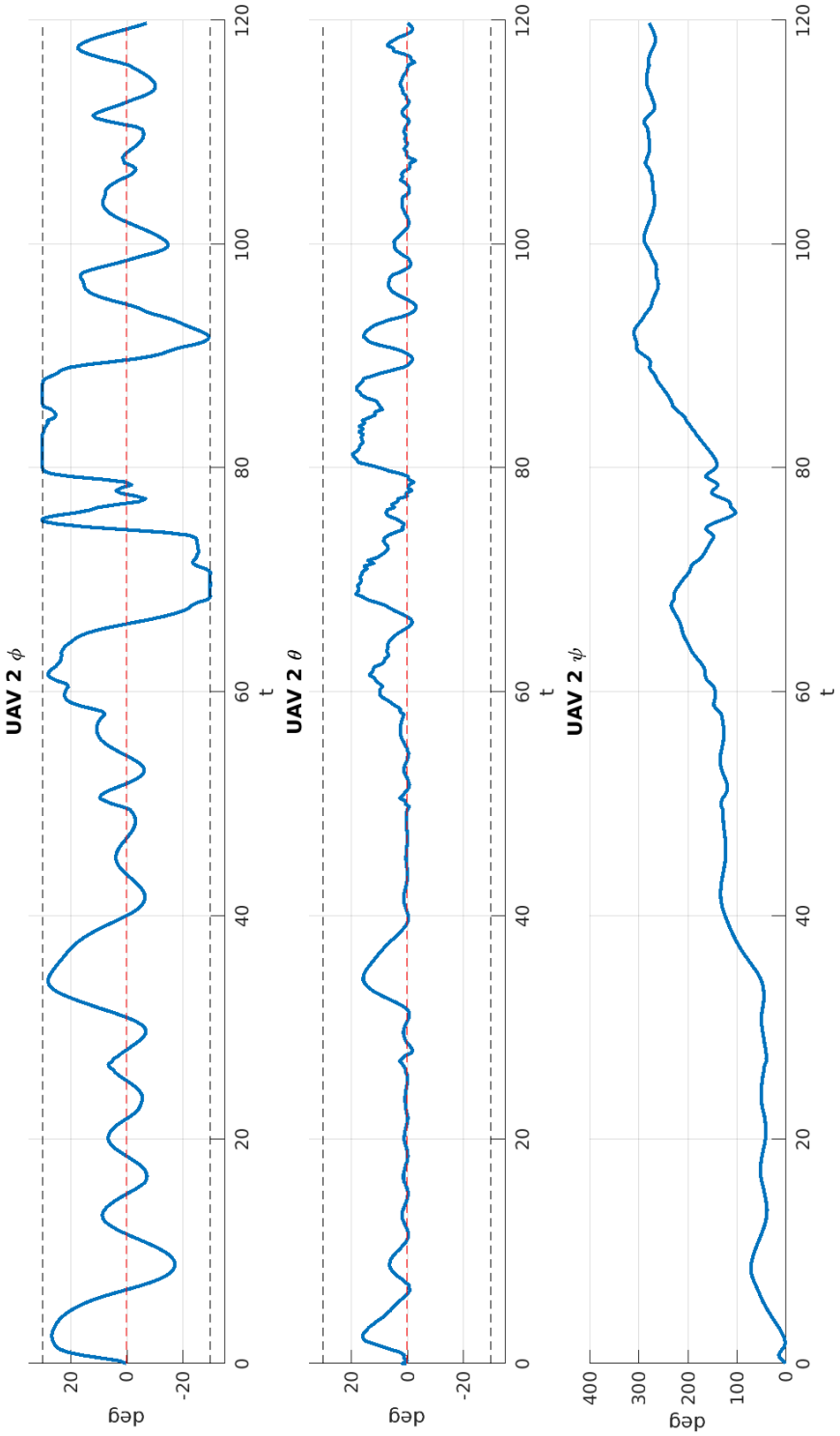


Figure 9.12: Data from the dynamic node position simulation. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

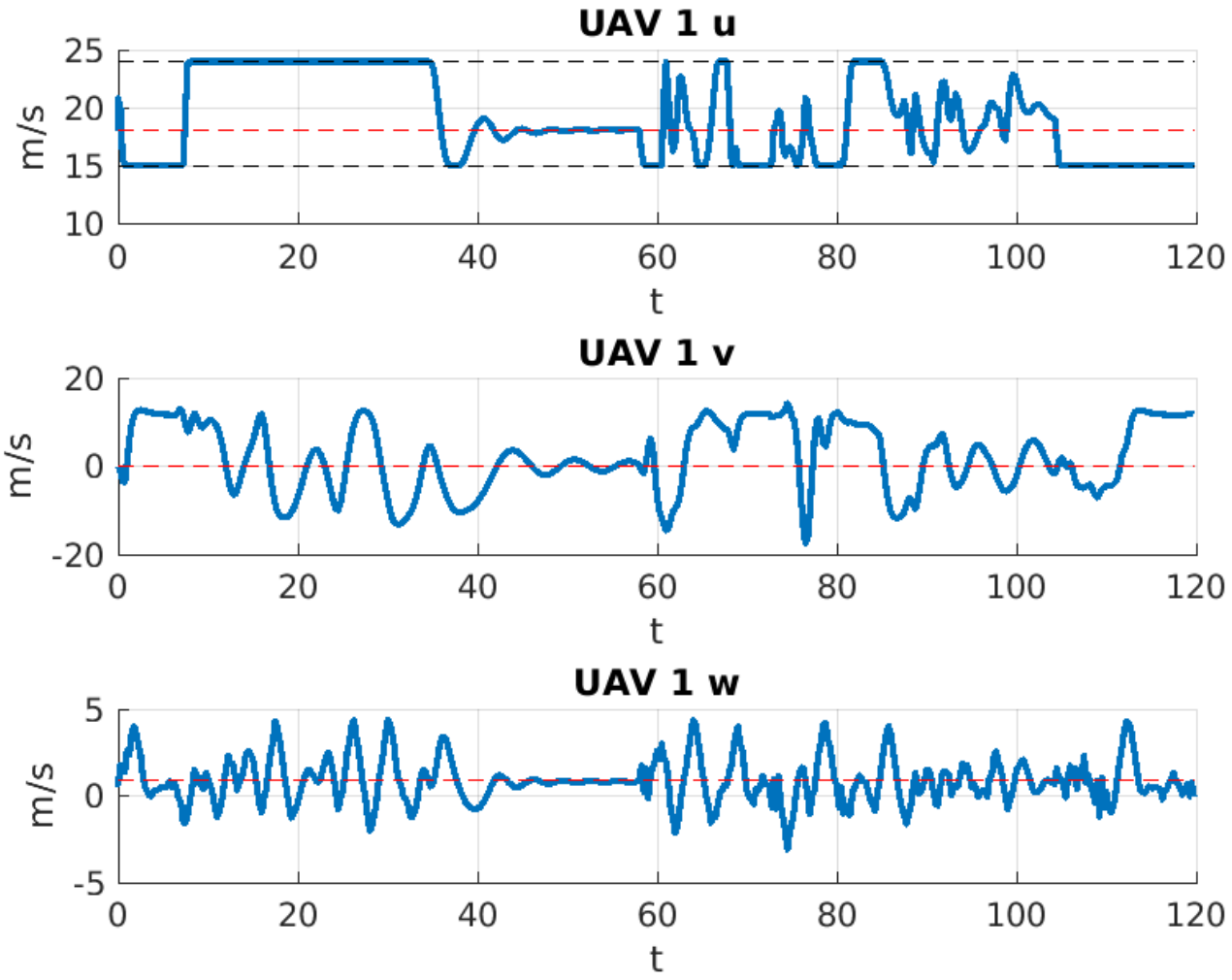


Figure 9.13: Data from the dynamic node position simulation. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

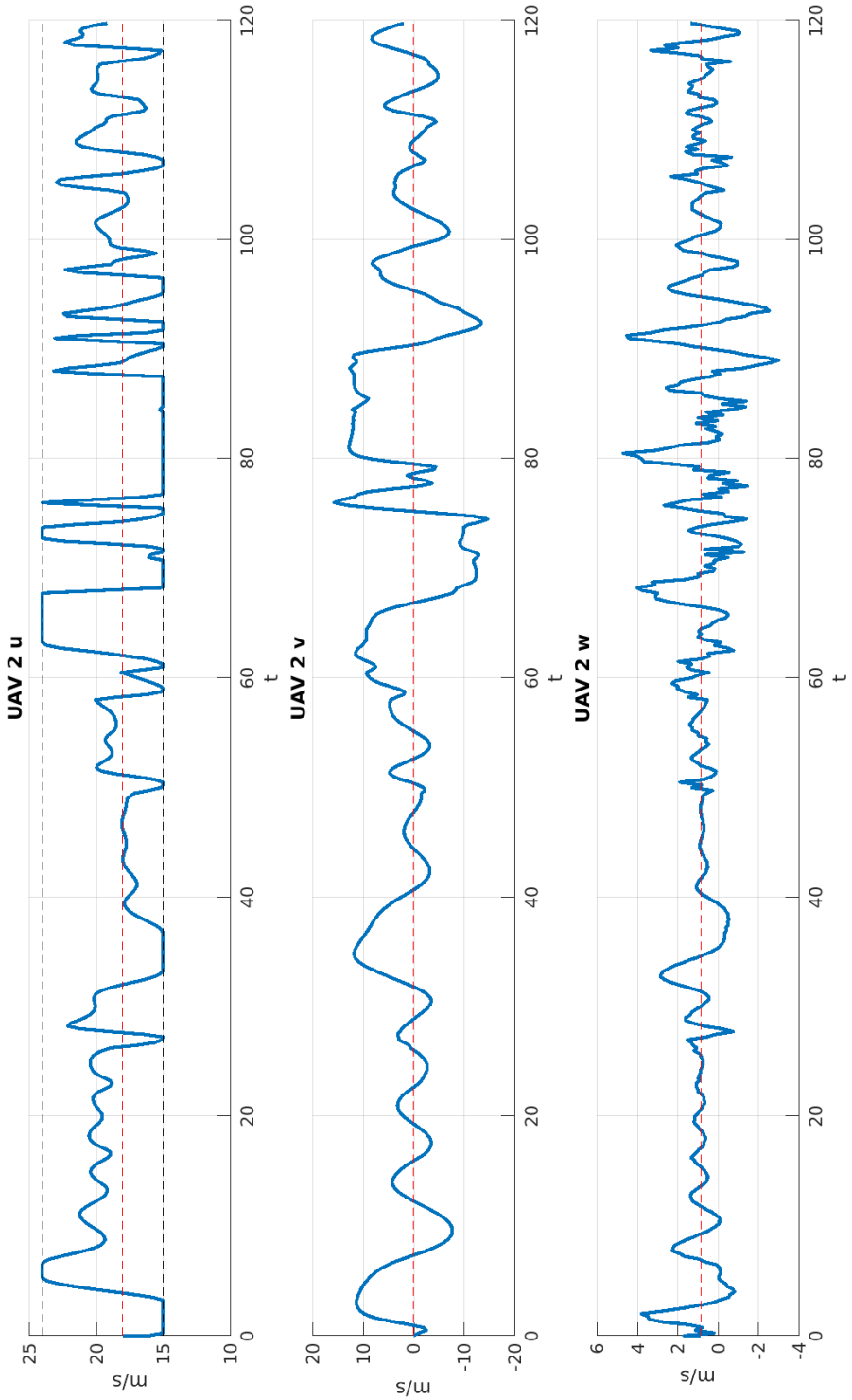


Figure 9.14: Data from the dynamic node position simulation. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

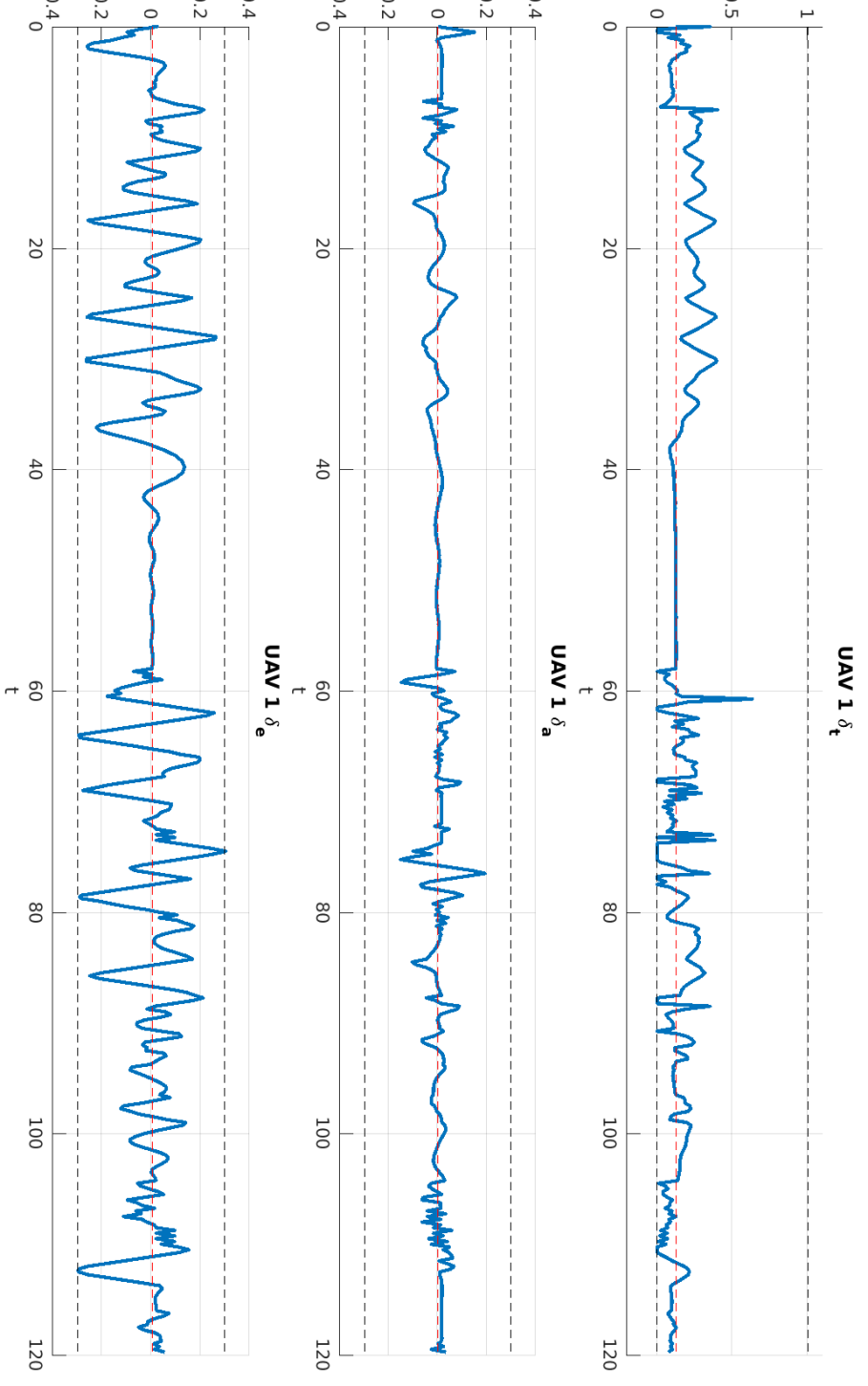


Figure 9.15: Data from the dynamic node position simulation. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

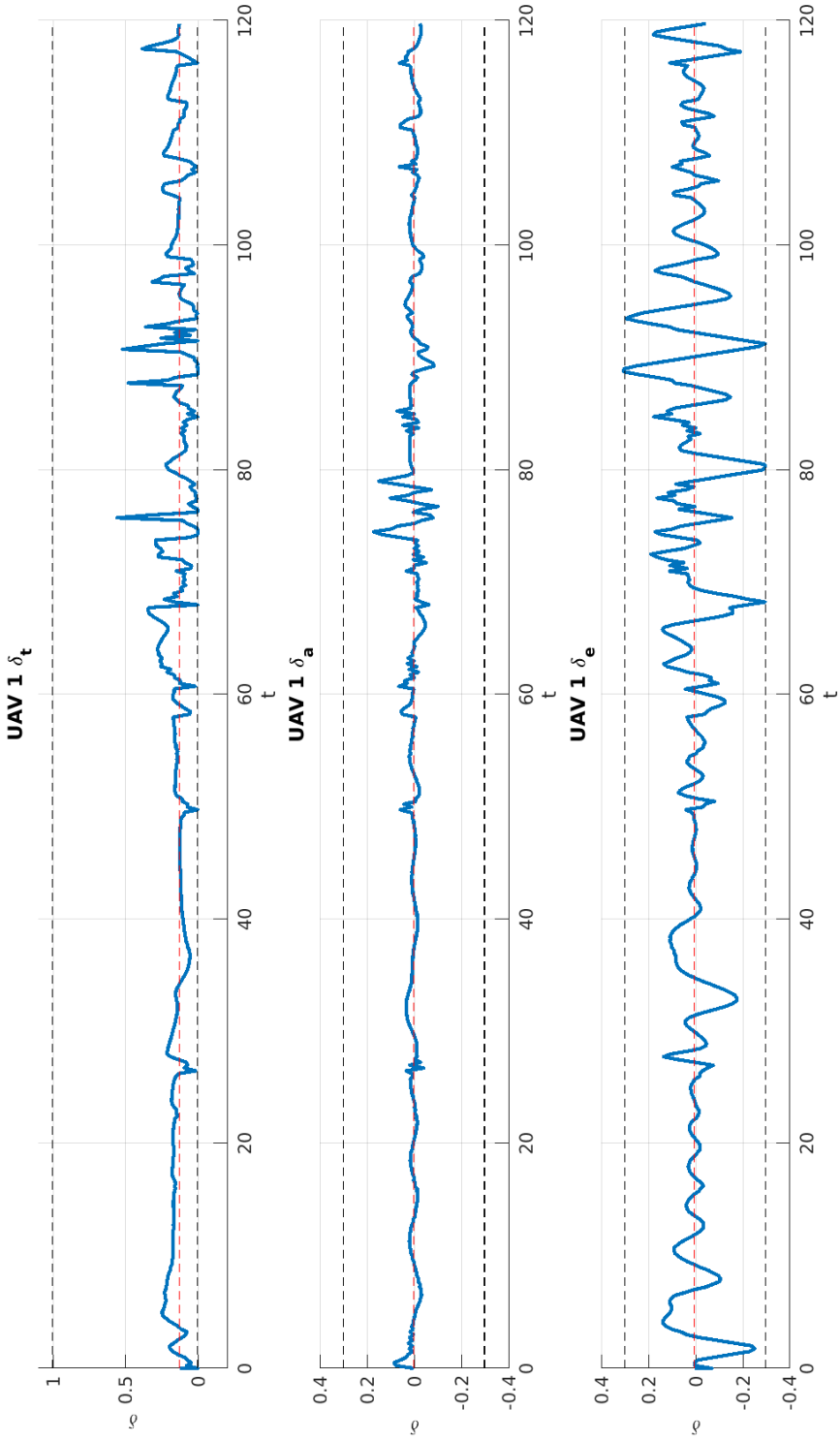


Figure 9.16: Data from the dynamic node position simulation. The dotted red line illustrates the trim conditions, and the black lines illustrates the constraints of the MPC.

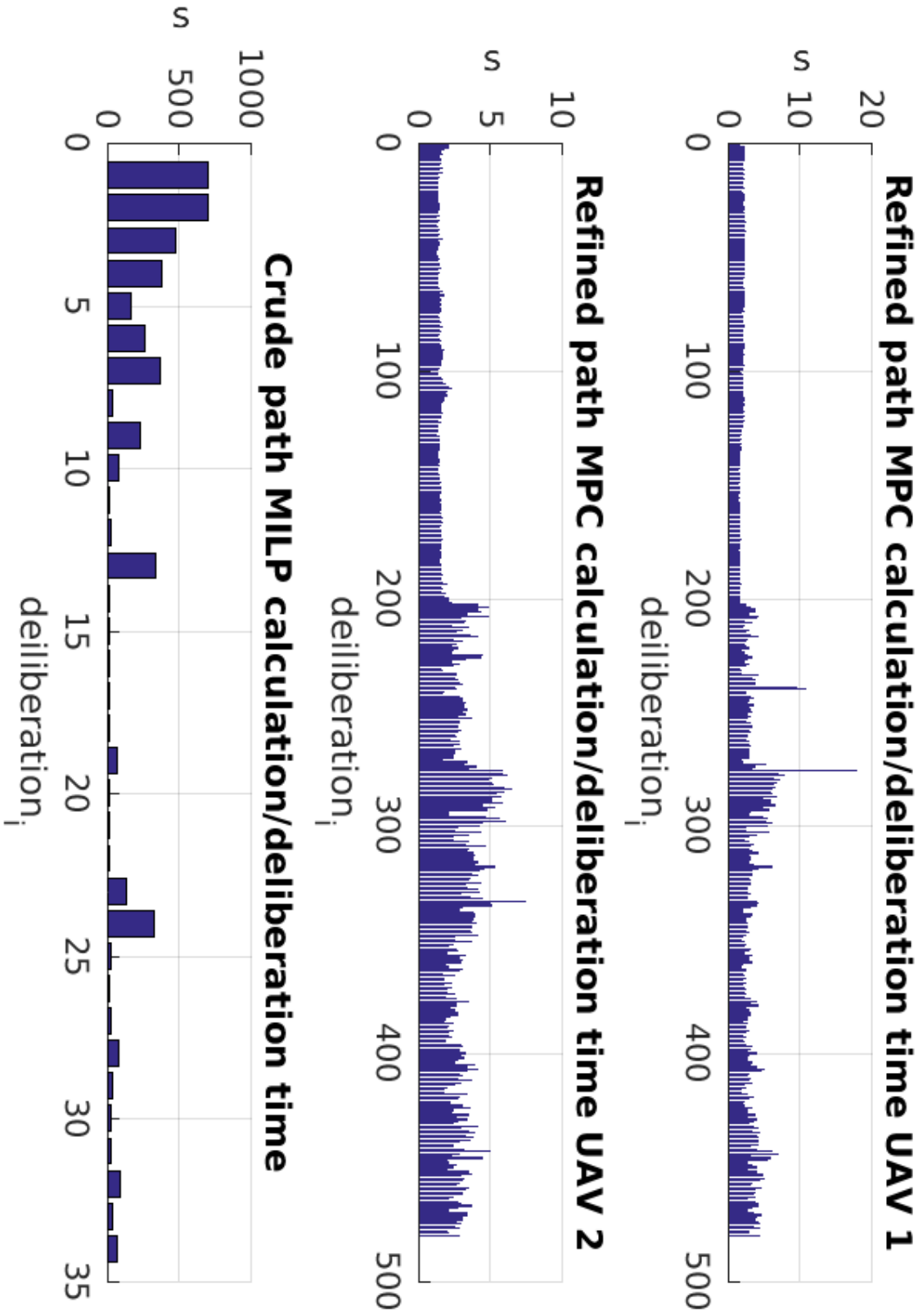


Figure 9.17: Data from the dynamic node position simulation.

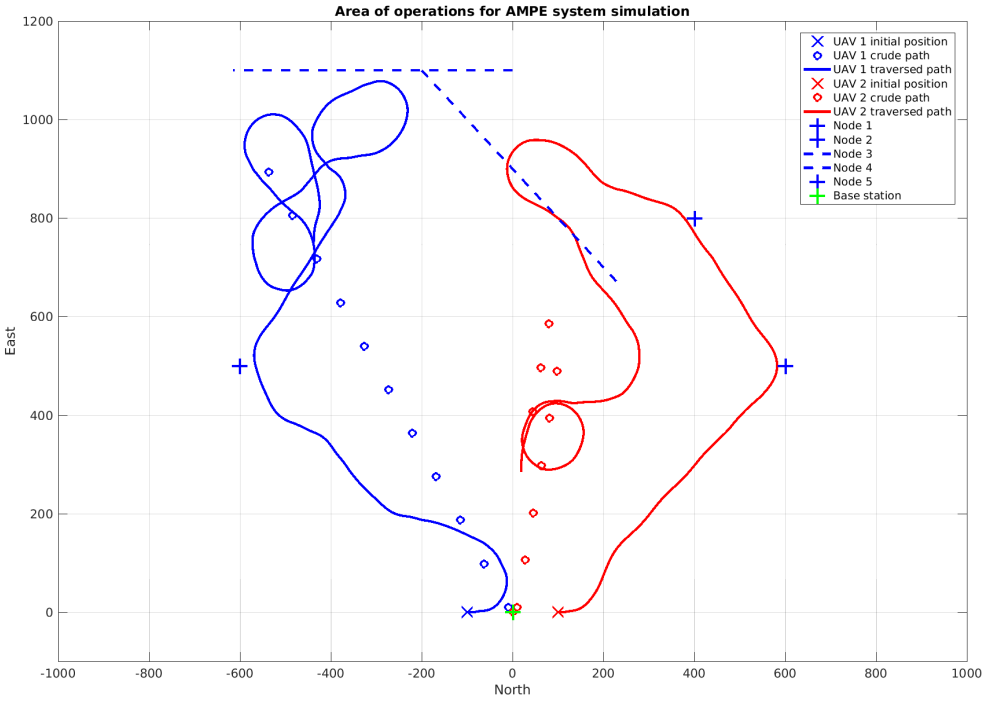


Figure 9.18: An unsuccessful mission due to large delay from the crude planning starts till it ends.

Part IV

Discussion and conclusion

CHAPTER 10

Discussion

10.1 MILP performance

The Mixed Integer Linear Programming model is shown to perform well in the sense of creating good paths, but long and unpredictable calculation times can rise cause to effects like shown in section 9.2. In addition, the deliberation time of the module can stay at 500 seconds for the first part of the mission. During this part the plan can stay static for 8 minutes. Most importantly, it can cause the UAV to miss the nodes, but the plan might also not be optimal any longer after so long. This diminishes the usefulness of having complex computation, with its trade-off computation times. As an observation, the MILP module seems to get significantly worse the more time steps that is required to complete the mission, while as the number of time steps is less important of both UAVs reach the base station within the time horizon.

The computation times shown in the results are without the data flow constraints. This also impacts the applicability of the MILP module. If the UAVs are to perform long missions, visiting a large amount of nodes before completion, the buffer size would not be enough. The scalabillity of the system suffers from this, and the observation that the longer it takes to complete the mission, the longer the deliberation. However, the sampling time can be increased to allow for a longer mission. Then the UAVs could reach further and therefore the sampling time must be scaled with the size of the mission area. This must be done with

care, not to risk loosing collision avoidance. UAV collision avoidance unreliable at best in mid flight because of the spread in the positions at each time step, but it is not important as UAVs travels to different nodes, and is only likely to get close at these nodes, and therefore only if approaching the end node at the same angle. If anti-grounding constraints are to be included however, the security of the UAV could suffer from increased time steps.

10.2 EUROPA performance

The EUROPA module was run in the static mission simulation, but the results points towards the module to be more suited for the dynamic mission. The planner chose a short path for the UAV 2 and had UAV handle all others. In this simulation, it is easy to tweak the temporal goals for this special case in order to get good results, but this would be counterproductive towards a general and autonomous system. Therefore the EUROPA simulation is done with the results given without any special configuration for the specific mission. The module, in difference from the MILP reactor, deliberates the plan quickly at only 1.8 seconds. This would make it suitable for dynamic missions, if the loss of optimally is acceptable. To accommodate both optimality and efficiency a combination of the two modules will be suggested later in this chapter.

10.3 MPC performance

The system shows very good results for following the paths in the static mission case, and performs stable control with small perturbation on the actuators. However, these results are found using the linear model, which are the same as used internally by the MPC. Good results are to be expected, and the work done on a external simulator shows that the results are not that simple in an real-time operation where the modelled flight dynamics are not perfect.

The MPC can react poorly when receiving a new plan. This can be seen for example at the end of the moving nodes simulation. When approaching the base station the MILP problem becomes trivial and is solved for every tick, giving a constant stream of new plans. At this approach, the UAV controls becomes somewhat ”shaky”. This seems to be because the intervals of the new references is skewed, causing the UAV to try and catch up or slow down to synchronize with the new trajectory. This is a challenge when dealing with parallel asynchronous systems, (until synchronized by the T-REX agent). The reaction of the MPC when introduced to this skewed input intervals is somewhat extreme, and causes unwanted deviations from the plan. Also, the changes in plan during that mission, though drastic, should

still be met with a smooth transition. A way to relax the reaction could be to introduce slew-rate to the objective function, as described in [27]. The idea is to use the previous output sequence over the planed time horizon as a input, and minimize the difference towards the new output (except the last output). This helps to introduce a soft transition in results from one MPC step to the next, and would be well suited as the change of plans in the AMPE system is a abrupt change in most input variables of the MPC. The slew-rate could help limit the effect of this disruption.

While T-REX is built with synchronization in mind as one of the main goals, it proves challenging to synchronize signals accurate in the discrete time intervals. There are some delays in the dispatching of goals, however, this is deterministic and one should be able to compensate this. The challenge here seems to be that the planned speed is given to the UAV as timed reference signal of position. When the new plan has a slightly skewed time horizon, the MPC will try to reach that waypoint sooner or later than intended. It will have to catch up or slow down to synchronize with the new phase of the intervals. This could be solved with some heuristics or a interpolation between the old and the new reference trajectories when a new plan is complete.

A significant issue with the MPC in its current state is, along with the MILP module, the computation time of each step. Averaging between one and two seconds is too slow for the fast UAV dynamics. ACADO provides a tool which automatically generates optimized MPC code in C++ from Matlab code, and this is suggested as a way of reducing the computation times. This code would be optimized for the specific program at hand.

10.4 Applicability in real-time operations

The AMPE system have been shown to be able to perform optimal planning, follow a given and changing trajectory and achieving smooth and precise maneuvers. It is important to remember that this is in simulations that is not performed in real-time. The planning time of the MILP crude path planner is unpredictable and have sometimes exceeded five minutes. In this case, the mission would be completed before the plan was finished and all mission dynamics would be lost. The deliberations of each individual module would be shorter on the intended distributed system. In these simulations, the crude path planner, and both MPCs are running at the same processor, while as in a deployment of the AMPE system, one agent would be on the ground responsible for the crude planning of the overall mission, while each MPC would mainly perform only one MPC task, or a MPC and PID task. This would lighten

the load significantly, but it would require expensive computational equipment aboard the UAVs. This is not desirable as the loss of a UAV would be more expensive. Still, some required processing power aboard the UAV must be expected, and sensors might be much more costly anyways. The ground station could keep powerful computers, for the common agent. This can be impractical with mobile base stations at land, but for a base aboard a vessel, A permanent setup able to do the MILP planning in a timely manner might be a good solution.

For real time demands using the available equipment for simulations, an alternative setup was tested when working with the system at an external simulator. By using the MPC to determine waypoints given to a PID regulator, the system was able to keep stable control in the initial testing. This gives a smoother path and avoids 180 degree turns which occurs in the crude plans. A Dubins path wold be much less costly to calculate and would remedy the same issues, which leaves the usefulness of such a setup in question. There are still some advantages of the MPC planning approach. When data flow constraints are used, the crude plan is determined by time stamped waypoints, and if the UAVs are to meet for data transfer they are dependent on keeping these temporal goals. The MPC does not only optimize the path to be the shortest as Dubins path does [7], but also optimizes the time it is at the waypoint, changing the path accordingly.

This thesis have focus on the use of both Artificial Intelligence and Mathematical Optimization to built the system. The system is built into an AI framework with the architecture of a continuous plan implemented on timelines by T-REX. For the modules implemented with MILP optimization, the results give good paths, but the unpredictable makes it risky for real operations. This is where there can be great compatibility between the EUROPA AI module and the MILP module. The EUROPA module performed its task in just about 1.8 seconds. This is will within the limits of what is needed of the crude path planner. Therefore, for a real system a combination of these two systems could be favorable. The MILP would perform the initial plan, which is then followed up by the EUROPA module until the MILP module completes a new plan. EUROPA have the feature of rejectable goals, meaning the goals would only apply if they are possible to reach. When receiving the MILP plan the temporal relations between the visit of each node could be set as rejectable goals, causing EUROPA to change from the initially optimal order of nodes to visit only when it would be unfeasible. This would introduce more optimally to the EUROPA reactor. It would be useful to simulate the system with online deliberation of EUROPA to test this theory, and therefore it is left to future work to get in contact with the NASA EUROPA team for possible work-arounds of

the compilation issues.

10.5 Notes on stability

A MPC with unconstrained actuators are guaranteed to be stable [21], however this is not realistic in the real world and all controls of the X8 UAV must be constrained to the saturation. This makes the problem of stability complex, ... It becomes even more complex as the Runge-Kutta method used in the MPC can become unstable, rendered unable to find a step length for which the requested accuracy is achieved, at very rapid dynamics. The constrains added to the MPC should prevent such scenarios, but if the MPC is not stable it might not be able to keep these constrains. [21] goes into detail on the study of closed-loop stability when using Model Predictive Controls. As the AMPE system have yet to produce any stable results on external systems, this is left for future work. The first challenge of the control in the AMPE system would be to reduce the computation time of the MPC if it is to be used as a controller.

10.6 A combined system of MILP and EUROPA

A module combining Europa and MILP path planning will be suggested here. At the start of the mission, spending ten minutes at deliberating a plan does not affect the mission success. During the flight it is more important to keep the deliberation time down, or have some other technique to get intermediate plans between each deliberation. This is how the two concepts would work together. The MILP module would provide a initial plan, and set the planned approach of nodes as goals for the EUROPA reactor. EUROPA features as mentioned rejectable goals. The temporal relations between node visits from the MILP plan can be set as rejectable goals (before, after...). Then EUROPA would follow the plan of the MILP reactor while being able to use much fresher position samplings of the nodes for the plan.

CHAPTER 11

Future work

This system, while considering a variety of conditions such as resources, and for static missions the data flow through the network, still faces some simplifications. A simple quadratic function was used for the path loss, but in [18] a solution is presented using SPLAT! to check whether the connectivity holds at the a positions where the UAVs attempts a data transfer. This is done by iteratively checking a plan to see if the requested path loss actually holds, and if not constrain R^{con} more. This might be impractical as it potentially multiplies the deliberation time of the module several times. The online usage is already strained as shown by the logged times in the results, but in a scheme using both EUROPA and MILP, it could become feasible as a means to ensure the data transfer is possible. Another simplification that were touched upon is the off-shore usage of the AMPE system. In order to execute mission above land, some form of terrain avoidance must be implemented. This is done in [17],

The AMPE system is in this thesis developed with a specific case in mind, which is that of the data gathering be establishing a relay network or ferrying data to a base station. The T-REX system is built to allow hierarchical planning, and in order to generalize the system to more cases than the one specified here, another level of planning can be built on top of the existing system. Given a working combination of EUROPA and MILP for crude path planning, all capabilities of the UAV using the AMPE system can be contained as building blocks. Then the new planning level will utilize these building blocks to perform a mission

within constraints and parameters given. Say or example the data flow constraints. If the UAV was only searching for drifting ships as a default, the data flow constrains could be added as a block to include data transmissions with any found ship.

Finally, stability analysis, adding and tuning slew rate objectives to the MPC and to utilize the ACADO code generation tool for a optimized MPC have all been discussed before as future work.

CHAPTER 12

Conclusion

This thesis sat out to develop a mission planning and execution system by using T-REX with its AI approach as inspiration. The system should be autonomous and subtasks of this system were to be tested using EUROPA together with Mathematical optimization. The resulting AMPE system where developed to test the T-REX system in which some or all modules were replaced from EUROPA planners to MPC and MILP modules. To this end the results have shown different tendencies towards applicability of each of these modules. The overall system architecture have proven smart for the optimization methods. The parallel design suits the slow deliberation of the MILP module, and the architecture of using timelines for each module to fill with tokens, iterated through as the time progresses makes it easy for the MPC to collect the current goals an insert it into its own planning horizon. Both the MILP model and the MPC uses the receding time horizon principle in this thesis, giving then naturally a planning horizon which correlates well with the T-REX functionality. The Mathematical Optimization's had been even more suited had the observations received been included directly into the current planning, but this is not applicable for these methods.

Both the MPC and the MILP modules have proven to be too slow for any online use in its current configuration. The MPC can be improved to reach computation time that allows for online control by taking measures such as utilizing the ACADO code generation tools which optimizes the controller for the specific program used. The MILP module does not seem to have a potential of being efficient enough to be used as a standalone online planner.

The MILP module have been shown to sometimes use more than five minutes to reach an optimal plan, but this plan is guaranteed to be optimal within some limit. However, a static mission have been planed by the EUROPA module, showing a successful mission with much faster deliberation time than the MILP module. The ability to test the Artificial Intelligence in the form of the crude path planning reactor implemented with an EUROPA module, have been less than desirable. However, the results that have been established tends towards both a problem with the MILP module, and a solution using EUROPA. The extensive delays in the MILP path planning introduces a risk of the system not completing the mission if the dynamics are to great. A case was shown where 4 nodes where moving, at which the delay of the MILP plan resulted in the UAVs always missing the nodes.

In the end, the system is a functional autonomous system, which takes in only the node positions and plans the entire mission, then executing it while re-planing in order to maintain that plan, but which is yet not able to meet any real-time demands. The performance of the optimization modules are less then desirable, and therefore improvements and designs including EUROPA have been suggested. The concepts of T-REX, EUROPA, MILP and MPC shows good compatibility, and further development by combining the efficiency of EUROPA with the optimality of the MILP module have potential for a functional online mission planning module allowing for real-time autonomous mission planning and execution.

CHAPTER 13

Bibliography

[1] How does europa represent and think about plans? https://github.com/nasa/europa/wiki/Planning-Approach. Accessed: 2017-02-20.

[2] How to build head of master. https://github.com/nasa/europa/issues/172. Accessed: 2017-0-25.

[3] Mathworks (2015a). airframe trim and linearize. http://se.mathworks.com/help/aeroblks/examples/airframe-trim-and-linearize-html#zmw57dd0e1168. Accessed: 2017-06-22.

[4] Nddl reference manual. https://github.com/nasa/europa/wiki/NDDL-Reference. Accessed: 2017-02-20.

[5] A simple sample application (the planetary rover). https://github.com/nasa/europa/wiki/Example-Rover. Accessed: 2017-04-13.

[6] Ali Ahmadzadeh, Ali Jadbabaie, Vijay Kumar, and George J. Pappas. Multi-uav cooperative surveillance with spatio-temporal specifications. pages 5293–5298, 2006.

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

[8] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[9] Eduardo F. Camacho, Carlos Bordons Alba, and SpringerLink. *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer London : Imprint: Springer, 2nd ed. edition, 2007.

[10] James. Cary, Leslie; Coyne. Icao unmanned aircraft systems (uas), circular 328". 2011-2012 uas yearbook - uas: The global perspective. *Blyenburgh Co.*, page pp. 112–115, 2012.

[11] A. Chaudhry, K. Misovec, and R. D' Andrea. Low observability path planning for an unmanned air vehicle using mixed integer linear programming. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 4, pages 3823–3829 Vol.4.

[12] Olav Egeland and Jan Tommy Gravdahl. *Modeling and simulation for automatic control*. Marine Cybernetics, Trondheim, 2002.

[13] Bjarne Foss and Aksel N. Heirung. Merging optimization and control. 2014.

[14] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Vademecum de navium motu contra aquas et de motu gubernando. Wiley, New York, 2011.

[15] F. Gavilan, R. Vazquez, and E. F. Camacho. An iterative model predictive control algorithm for uav guidance. *IEEE Transactions on Aerospace and Electronic Systems*, 51(3):2406–2419, 2015.

[16] Kristoffer Gryte and Thor Inge Fossen. *High Angle of Attack Landing of an Unmanned Aerial Vehicle*. Thesis, 2015.

[17] Esten I. Grøtli and Tor A. Johansen. Motion- and communication-planning of unmanned aerial vehicles in delay tolerant network using mixed-integer linear programming. *Modeling, Identification and Control*, 37(2):77–97, 2016.

[18] Esten Ingar Grøtli and Tor Arne Johansen. Path planning for uavs under communication constraints using splat! and milp. *Journal of Intelligent Robotic Systems*, 65(1):265–282, 2012.

[19] B. Houska, H.J. Ferreau, M. Vukov, and R. Quirynen. ACADO Toolkit User's Manual. http://www.acadotoolkit.org, 2009–2013. Accessed: 2017-05-30.

[20] D. Koks. Using rotations to build aerospace coordinate systems, australian government electronic warfare and radar division systems sciences laboratory. *DSTO*, 2008.

[21] Basil Kouvaritakis and Mark Cannon. *Nonlinear predictive control : theory and practice*, volume 61 of *IEE control engineering series*. Institution of Electrical Engineers, London, 2001.

[22] Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen. T-rex: A model-based architecture for auv control. In *3rd Workshop on Planning and Plan Execution for Real-World Systems*, volume 2007.

[23] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer series in operation research and financial engineering. Springer, New York, 2nd ed. edition, 2006.

[24] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl. Autogenerating microsecond solvers for nonlinear mpc: A tutorial using acado integrators. *Optimal Control Applications and Methods*, 36(5):685–704, 2015.

[25] AJM Raemaekers. Design of a model predictive controller to control uavs. Report, Technische Universiteit Eindhoven, 2007 2007.

[26] Kanna Rajan, Frédéric Py, and Javier Barreiro. *Towards deliberative control in marine robotics*, pages 91–175. Springer New York, 2013.

[27] Thomas J. Stastny, Adyasha Dash, and Roland Siegwart. *Nonlinear MPC for Fixed-wing UAV Trajectory Tracking: Implementation and Flight Experiments*. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, 2017.

[28] Henning A. Åsgård. Combining methods of mathematical optimization and artificial intelligence for autonomous uav mission planning, execution and resource management. 2016.

Part V

Appendix

APPENDIX A

UAV model constants and parameters

This chapter will provide the remaining constants and parameters for the UAV model illustrated in chapter 2. This is a similar to the table in [7].

A.1 UAV linear model constants

Lateral	Formula
Y_v	$\frac{\rho S b v^*}{4mV_a^*}[C_{Y_p}p^* + C_{Y_r}r^*] + \frac{\rho S v^*}{m}[C_{Y_0} + C_{Y_\beta}\beta^* + C_{Y_{\delta_a}}\delta_a^* + C_{Y_{\delta_r}}\delta_r^*] + \frac{\rho S C_{Y_\beta}}{2m}\sqrt{u^{*2} + w^{*2}}$
Y_p	$w^* + \frac{\rho V_a^* S b}{4m}C_{Y_p}$
Y_r	$-u^* + \frac{\rho V_a^* S b}{4m}C_{Y_p}$
Y_{δ_a}	$\frac{\rho V_a^{*2} S}{4m}C_{Y_{\delta_a}}$
Y_{δ_r}	$\frac{\rho V_a^{*2} S}{4m}C_{Y_{\delta_r}}$
L_v	$\frac{\rho S b v^{*2}}{4mV_a^*}[C_{p_p}p^* + C_{p_r}r^*] + \rho S v^* b[C_{p_0} + C_{p_\beta}\beta^* + C_{p_{\delta_a}}\delta_a^* + C_{p_{\delta_r}}\delta_r^*] + \frac{\rho S C_{p_\beta}}{2}\sqrt{u^{*2} + w^{*2}}$
L_p	$\Gamma_1 q^* + \frac{\rho V_a^* S b^2}{4}C_{p_p}$
L_r	$\Gamma_2 q^* + \frac{\rho V_a^* S b^2}{4}C_{p_r}$
L_{δ_a}	$\frac{\rho V_a^* S b^2}{2}C_{r_{\delta_a}}$
L_{δ_r}	$\frac{\rho V_a^* S b^2}{2}C_{r_{\delta_r}}$
N_v	$\frac{\rho S b v^{*2}}{4mV_a^*}[C_{p_p}p^* + C_{p_r}r^*] + \rho S v^* b[C_{r_0} + C_{r_\beta}\beta^* + C_{r_{\delta_a}}\delta_a^* + C_{r_{\delta_r}}\delta_r^*] + \frac{\rho S C_{r_\beta}}{2}\sqrt{u^{*2} + w^{*2}}$
N_p	$\Gamma_7 q^* + \frac{\rho V_a^* S b^2}{4}C_{r_p}$
N_r	$-\Gamma_1 q^* + \frac{\rho V_a^* S b^2}{4}C_{r_r}$
N_{δ_a}	$\frac{\rho V_a^* S b^2}{4}C_{p_{\delta_a}}$
N_{δ_r}	$\frac{\rho V_a^* S b^2}{4}C_{p_{\delta_r}}$

Longitudinal	Formula
X_u	$\frac{\rho S w^* C_{X_\alpha}}{2mV_a^*} + \frac{\rho S u^*}{m}[C_{X_0} + C_{X_\alpha}\alpha^* + C_{X_{\delta_e}}\delta_e^*] + \frac{\rho S c C_{X_q} u^* q^*}{4mV_a^*} - \frac{\rho S_{prop} C_{prop} u^*}{m}$
Y_p	$-q + \frac{\rho S w^*}{m}[C_{X_0} + C_{X_\alpha}\alpha^* + C_{X_{\delta_e}}\delta_e^*] + \frac{\rho S c C_{X_q} w^* q^*}{4mV_a^*} + \frac{\rho S u^* C_{X_\alpha}}{2mV_a^*} - \frac{\rho S_{prop} C_{prop} w^*}{m}$
X_q	$-w^* + \frac{\rho V_a^* S C_{X_q} c}{4m}$
X_{δ_e}	$\frac{\rho V_a^* S C_{X_{\delta_e}} c}{2m}$
X_{δ_t}	$\frac{\rho S_{prop} C_{prop} k^2 \delta_t^*}{m}$
Z_u	$q + \frac{\rho S w^* C_{Z_\alpha}}{2mV_a^*} + \frac{\rho S u^*}{m}[C_{Z_0} + C_{Z_\alpha}\alpha^* + C_{Z_{\delta_e}}\delta_e^*] + \frac{\rho S c C_{X_q} u^* q^*}{4mV_a^*}$
Z_w	$\frac{\rho S w^*}{m}[C_{Z_0} + C_{Z_\alpha}\alpha^* + C_{Z_{\delta_e}}\delta_e^*] + \frac{\rho S c C_{Z_q} w^* q^*}{4mV_a^*} + \frac{\rho S u^* C_{X_\alpha}}{2mV_a^*}$
Z_q	$-w^* + \frac{\rho V_a^* S C_{X_q} c}{4m}$
Z_{δ_a}	$\frac{\rho V_a^* S C_{Z_{\delta_e}} c}{2m}$
M_u	$\frac{\rho S w^* C_{X_\alpha}}{J_y} + \frac{\rho S u^*}{m}[C_{X_0} + C_{X_\alpha}\alpha^* + C_{X_{\delta_e}}\delta_e^*] + \frac{\rho S c C_{X_q} u^* q^*}{4mV_a^*}$
M_w	$\frac{\rho S w^* C_{Z_\alpha}}{2mV_a^*} + \frac{\rho S u^*}{m}[C_{M_0} + C_{M_\alpha}\alpha^* + C_{m_{\delta_e}}\delta_e^*] + \frac{\rho S c C_{m_q} u^* q^*}{4J_y V_a^*}$
M_q	$\frac{\rho V_a^* S C_{M_{\delta_e}} c}{4J_y}$
M_{δ_a}	$\frac{\rho V_a^* S C_{m_{\delta_e}} c}{4J_y}$

Parameters	
Longitudinal	Formula
C_i	Aerodynamic coefficients
ρ	Air density
k_{motor}	Efficiency of the motor
b	Wing span
α	Angle of attack
β	Side slip
ρ	air
ρ	air
ρ	air

A.2 The X8 flying-wing parameters

m 3.364
J_x 1.2289999886841
J_y 0.170199999115332
J_z 0.880799992835993
J_xz 0.934299991564547
S 0.75
b 2.1
c 0.357142858
S_prop 0.3556
rho 1.2682
k_motor 80
k_T_p 0
k_Omega 0
e 0.993384556116078

C_L_0 0.025906250396833
C_D_0
C_m_0 0.017293940631080
C_L_alpha 4.016155154888014
C_D_alpha
C_m_alpha −0.283327228485977
C_L_q 3.918552496580028

```
C_D_q 0
C_m_q -1.304655882352941
C_L_delta_e 0.587243508940072
C_D_delta_e 0.846082348709986
C_m_delta_e -0.485680915663958
C_prop 1.11
M 50
alpha_0
epsilon 0.1592
C_D_p 0.010182
C_n_delta_r 0

C_Y_0 0
C_l_0 0
C_n_0 0
C_Y_beta -0.194933999410735
C_l_beta -0.076488773525576
C_n_beta 0.026884995608090
C_Y_p -0.117176976744186
C_l_p -0.401796046511628
C_n_p -0.024744093023256
C_Y_r 0.095932174418605
C_l_r 0.024960209302326
C_n_r -0.125246744186047
C_Y_delta_a -0.069648368873249
C_l_delta_a 0.298694319687316
C_n_delta_a 0.007569499616129
C_Y_delta_r 0
C_l_delta_r 0
```

A.3 The X8 flying-wing trim condition

```
uStar 18.0168;
vStar 0.0001;
wStar 0.8366;
thetaStar (2.65868 / 180) * 3.14159265359;
```

```
phiStar 0.0;
psiStar 0.0;
pStar 0.0;
qStar 0.0;
rStar 0.0;
alphaStar 0.0;
betaStar 0.0;
delta_aStar 0.0000;
delta_rStar 0.0;
delta_eStar 0.0039;//0.0079;
delta_tStar 0.1240;
V_aStar 19.0;
hStar 0.0;
```

APPENDIX B

EUROPA nddl model

```
//#include "Plasma.nddl"
class UAV;
class Navigator;
class Node;
class Data;
class DataContainerState;
class DataContainer;
class Location;
class BaseStation;

class Location
{
        float n;
        float e;
        float d;
        Location(float _n, float _e, float _d)
        {
                n = _n;
                e = _e;
```

```
                d = _d;
        }
}


class Data
{
        string name;
        //int size;

        Data(string _name)
        {
                name = _name;
        }
        Data()
        {
                name = "unspecified";
        }
}

class DataContainerState
        extends Timeline
{
        predicate HasData {}
        predicate IsTransmitting {}
        predicate IsEmpty {}
}

class TranscieverState{

        predicate FreePred {}
        predicate TransmittingPred {}
}

class DataContainer
{
```

```
        Data data;

        DataContainerState state;

        DataContainer(string dataName)
        {
                data = new Data(dataName);
                state = new DataContainerState();
        }
        DataContainer()
        {
                data = new Data();
                state = new DataContainerState();
        }
        DataContainer(Data _data)
        {
                data = _data;
                state = new DataContainerState();
        }
}


class BaseStation
{
        Location location;

        TranscieverState state;

        BaseStation(float n, float e, float d)
        {
                location = new Location(n,e,d);
                state = new TranscieverState();
        }

        predicate RecievedDataPred
        {
```

```
                Data data;
        }
}

class Navigator
        extends Timeline
{



        predicate Going
        {
                Location from;
                Location to;
        }

        predicate At
        {
                Location at;
        }
}

class Node
{

        string dataName;
        Location location;

        DataContainer dataContainer;

        Node(string _dataName, float n, float e, float d)
        {
                dataName = _dataName;
                dataContainer = new DataContainer(_dataName);
                location = new Location(n, e, d);
        }
```

```
        Node(float n, float e, float d)
        {
                dataName = "unspecified";
                dataContainer = new DataContainer();
                location = new Location(n, e, d);
        }

        Node(Location _location, DataContainer _dataContainer)
        {
                dataName = "unspecified";

                dataContainer = _dataContainer;
                location = _location;
        }

}

Battery
    extends Reservoar
{
 string profileType;

        Battery(float ic, float ll_min, float ll_max)
        {
                super(ic, ll_min, ll_max);
                profileType="IncrementalFlowProfile";
}
}


class UAV
{

        Navigator navigator;

        UAV()
```

```
        {
                navigator = new Navigator();
        }

        predicate HasData
        {
                Data data;
        }

        action GOAct
        {
                Location to;
        }

        action CollectDataAct
        {
                Data data;
                Node possibleNodes;
        }

        action SendDataToBaseAct
        {
                Data data;
                BaseStation base;
        }

        action MeetUAVAct
        {
            UAV meet;

        }


}

UAV::GOAct
```

```
{
        this.start >= 0;

        met_by(condition object.navigator.At _from);
        meets(effect object.navigator.At destination);
        eq(to, destination.at);

        equals(effect object.navigator.Going going);
        eq(going.from, _from.at);
        eq(going.to, destination.at);

        float dist;
        calcDistance(dist, _from.at.n, _from.at.e, destination.at.n, de
        duration <= dist;
        duration >= dist - 0.9999;
}


UAV::CollectDataAct
{
        eq(duration, 2);

        this.start >= 0;

        possibleNodes.dataContainer.data == this.data;

        contained_by(condition object.navigator.At currentLocation);
        eq(currentLocation.at, possibleNodes.location);

        met_by(condition possibleNodes.dataContainer.state.HasData);

        starts(effect HasData inputData);
        eq(inputData.data, data);

        equals(effect possibleNodes.dataContainer.state.IsTransmitting)
        meets(effect possibleNodes.dataContainer.state.IsEmpty);
```

```
}

UAV::SendDataToBaseAct
{
        duration == 2;

        met_by(condition HasData hasData);
        hasData.data == this.data;

        contained_by(condition object.navigator.At currentLocation);
        currentLocation.at == base.location;


        after(condition CollectData isCollected);
        isCollected.data == this.data;
        isCollected.possibleNodes.dataContainer.data == data;


        equals(effect base.state.TransmittingPred);
        meets(effect base.state.FreePred);

        starts(effect base.RecievedDataPred recieved);
        recieved.data == data;

}
```

APPENDIX C

AMPL model

```
param np;        #total number of vehicles

param N;         # optimization horizon

param deltat;    # Sampletime

param Vmax;
param Vmin;

set Dim := 1..3;                    #NED


param initPos1{i in Dim};
param initPos2{i in Dim};

param initPos{p in 1..np, i in Dim};

param x_min;
```

```
param x_max;
param y_min;
param y_max;
param h_min;
param h_max;

#test params:
param goal{i in Dim};
param goal2{i in 1..2, j in 1..4};  #row first then collumn

param Dvel;
param chix{k in 1..Dvel, l in 1..Dvel/2};
param chiy{k in 1..Dvel, l in 1..Dvel/2};
param chiz{k in 1..Dvel, l in 1..Dvel/2};

param alphavel;                 #alpha for velocity estimation
param Mvel;

param rweight{j in Dim};                                    #acceleration wei

param W;
param dwp;
param Mwp;

param waypoints{w in 1..W, j in Dim};  #--- actual waypoints.

param M_finnish;
param gamma_finnish;            # positive weight for J_finnish


param dx;       #
param dy;       #       -anti-collition safety distance
param dz;       #

param MCol; # Big M andti-collision
```

```
param MConDim; # big M for casting conectivityDimentional to bool
param Rcon;
param epsilon;
param MCon;

param wind{j in Dim};

#Dataflow

param cSensor;
param hBar;
param Cmax;
param CmaxOut;
param CmaxIn;

#Resources constraint
param battery1;
param battery2;



#-------- Vars

var v {p in 1..np, i in 0..N, row in Dim};              #velocity in ve
var V {p in 1..np, i in 0..N};                                  #absolu

var pos {p in 1..np+1, i in 0..N, row in Dim};

var Jacc;                                       #The penalizing var for
var wacc{p in 1..np, i in 0..N, j in Dim};

var theta{w in 1..W};
var theta_finnish{p in 1..np};
var eta_finnish;                        # max finnishing time
var  J_finnish;                                 # finnishing time penna
```

```
var x{p in 1..np, i in 0..N};            #
var y{p in 1..np, i in 0..N};            #                -Used to read in
var z{p in 1..np, i in 0..N};            #

var lambda_sensor{p in 1..np, i in 0..N, t in 1..W};

var UAV_distances{p in 1..np, q in 1..np, i in 0..N};


#Dataflow

var m{p in 1..np+1, i in 1..N, s in 1..np, j in 1..N };
var c{p in 1..np+1, q in 1..np+1, i in 1..N, s in 1..np, j in 1..N };


#--- binary vars

var bvel{p in 1..np, i in 0..N, k in 1..Dvel, l in 1..Dvel/2} binary;

var bwp{p in 1..np, i in 0..N, w in 1..W} binary;

var bsensor{p in 1..np, i in 0..N} binary;

var bCol{p in 1..np-1, q in 2..np, i in 0..N, row in 1..3, col in 1..2};

var lambdaCon{p in 1..np + 1, q in 1..np + 1, i in 1..N, d in 1..Dvel, j
var bCon{p in 1..np+1, q in 1..np+1, i in 0..N} binary;

#-------- test vars

var numberOf_lambdaCon{p in 1..np+1, q in 1..np+1, i in 1..N};

#-------- model

#objective goes here:
```

```
minimize objective: J_finnish + Jacc;

#constraints:

#initialPosition

subject to initPos1constraint{p in 1..np, r in Dim}:
pos[p,0,r] = initPos[p,r];

#Base station

subject to baseStation{i in 1..N, r in Dim}:
pos[np+1,i,r] = waypoints[W, r];

#subject to initPos2constraint{r in Dim}:
#pos[2,0,r] = initPos2[r];

#UAV model

subject to model{p in 1..np, i in 0..N-1, row in Dim}:
pos[p, i+1, row] = pos[p,i,row] + deltat * (v[p,i,row] + wind[row]);

#velocity constraints


subject to Vestimate1{p in 1..np, i in 0..N-1, k in 1..Dvel, l in 1..Dv
V[p,i] >= v[p,i,1] * chix[k,l] + v[p,i,2] * chiy[k,l] + v[p,i,3] * chiz

subject to Vestimate2{p in 1..np, i in 0..N-1, k in 1..Dvel, l in 1..Dv
alphavel*(v[p,i,1] * chix[k,l] + v[p,i,2] * chiy[k,l] + v[p,i,3] * chiz

subject to Vestimate3{p in 1..np, i in 0..N-1}:
sum{k in 1..Dvel, l in 1..Dvel/2}( bvel[p,i,k,l] ) = 1;

subject to velocityConstraintsMax{p in 1..np, i in 0..N}:
V[p,i] <= Vmax * (1 - bwp[p,i,W]);
```

```
subject to velocityConstraintsMin{p in 1..np, i in 0..N}:
V[p,i] >= Vmin * (1 - bwp[p,i,W]);


#position constraints

subject to posx1{p in 1..np, i in 0..N}:
pos[p,i,1] >= x_min;

subject to posx2{p in 1..np, i in 0..N}:
pos[p,i,1] <= x_max;

subject to posy1{p in 1..np, i in 0..N}:
pos[p,i,2] >= y_min;

subject to posy2{p in 1..np, i in 0..N}:
pos[p,i,2] <= y_max;

subject to posh1{p in 1..np, i in 0..N}:
pos[p,i,3] >= h_min;

subject to posh2{p in 1..np, i in 0..N}:
pos[p,i,3] <= h_max;


subject to accelerationConstraint1:
Jacc = sum{p in 1..np, i in 0..N-2, j in Dim}(rweight[j] * wacc[p,i,j]);

subject to accelerationConstraint2{p in 1..np,i in 0..N-2, j in Dim}:
(v[p,i, j] - v[p, i+1, j]) <= wacc[p, i, j];

subject to accelerationConstraint3{p in 1..np,i in 0..N-2, j in Dim}:
-(v[p,i, j] - v[p, i+1, j]) <= wacc[p, i, j];

#------ Task assignment
```

```
subject to waypointHitP{p in  1..np, i in 1..N, w in 1..W, j in Dim}:
pos[p,i,j] - waypoints[w,j] - dwp <= Mwp * (1 - bwp[p,i,w]);

subject to waypointHitN{p in  1..np, i in 1..N, w in 1..W, j in Dim}:
 - pos[p,i,j] + waypoints[w,j] - dwp <= Mwp * (1 - bwp[p,i,w]);

 subject to wpOnce{w in 1..W-1}:
 sum{p in 1..np, i in 1..N}(bwp[p,i,w]) = 1;

  subject to AllToLast{ p in 1..np}:
 sum{i in 1..N}(bwp[p,i,W]) >= 1;


 subject to bStayTrue{p in 1..np, i in 1..N-1, w in 1..W}:
 bwp[p, i+1, W] >= bwp[p,i,w];


 subject to timeBeforeWP{w in 1..W-1}:
 theta[w] = sum{p in 1..np, i in 1..N}(i*bwp[p,i,w]);


 subject to finnishTime1{p in 1..np, i in 0..N}:
 theta_finnish[p] <= M_finnish * (1 - bwp[p,i,W]) + i*bwp[p,i,W];

 subject to finnishTime2{p in 1..np, i in 0..N}:
 theta_finnish[p] >= (i + 1) * (1 - bwp[p,i,W]);

 subject to timePennalty1{p in 1..np}:
 eta_finnish >= theta_finnish[p];

 subject to timePennalty2:
 J_finnish = gamma_finnish * eta_finnish;
```

```
 #Set reactor readable vars

 subject to xPos{p in 1..np, i in 0..N}:
 x[p,i] = pos[p,i,1];
 subject to yPos{p in 1..np, i in 0..N}:
 y[p,i] = pos[p,i,2];
 subject to zPos{p in 1..np, i in 0..N}:
 z[p,i] = pos[p,i,3];

#anti collision constraints

subject to antiCollisionX1{p in 1..np-1, q in p+1..np, i in 1..N} :
pos[p,i,1] - pos[q,i,1] >= dx - MCol * bCol[p,q,i,1,1];
subject to antiCollisionX2{p in 1..np-1, q in p+1..np, i in 1..N} :
pos[p,i,1] - pos[q,i,1] <= MCol * bCol[p,q,i,1,2] - dx;

subject to antiCollisionY1{p in 1..np-1, q in p+1..np, i in 1..N} :
pos[p,i,2] - pos[q,i,2] >= dy - MCol * bCol[p,q,i,2,1];
subject to antiCollisionY2{p in 1..np-1, q in p+1..np, i in 1..N} :
pos[p,i,2] - pos[q,i,2] <= MCol * bCol[p,q,i,2,2] - dy;

subject to antiCollisionZ1{p in 1..np-1, q in p+1..np, i in 1..N} :
pos[p,i,3] - pos[q,i,3] >= dz - MCol * bCol[p,q,i,3,1];
subject to antiCollisionZ2{p in 1..np-1, q in p+1..np, i in 1..N} :
pos[p,i,3] - pos[q,i,3] <= MCol * bCol[p,q,i,3,2] - dz;

subject to antiCollisionSum{p in 1..np-1, q in p+1..np, i in 1..N} :
sum{row in 1..3, col in 1..2}(bCol[p,q,i,row,col]) <= 5;


 #Data gathering constraints

subject to gather1{p in 1..np, i in 1..N, t in 1..W-1}:
lambda_sensor[p,i,t] <= sum{k in 1..i}(bwp[p,k,t]);

subject to gather2{p in 1..np, i in 1..N, t in 1..W-1}:
```

```
lambda_sensor[p,i,t] <= sum{k in i..N}(bwp[p,k,t]);

subject to gather3{p in 1..np, t in 1..W-1}:
sum{i in 1..N}(lambda_sensor[p,i,t]) = sum{i in 1..N}(i * bwp[p,i,1]) -

subject to sensorActive{p in 1..np, i in 1..N}:
sum{t in 1..W}(lambda_sensor[p,i,t]) = bsensor[p,i];

#Dataflow constraints


#connectivity constraints

subject to connectivityLambda1X{p in 1..np + 1, q in 1..np+1, i in 1..N
chix[d, l] * (pos[p,i, 1] - pos[q,i,1] ) - Rcon <= MConDim * (1 - lambda

subject to connectivityLambda2X{p in 1..np + 1, q in 1..np+1, i in 1..N
chix[d, l] * (pos[p,i, 1] - pos[q,i,1] ) - Rcon >= epsilon + (-MConDim
- epsilon) * lambdaCon[p, q, i, d, l, 1];


subject to connectivityLambda1Y{p in 1..np + 1, q in 1..np+1, i in 1..N
chiy[d, l] * (pos[p,i, 2] - pos[q,i,2] ) - Rcon <= MConDim * (1 - lambda

subject to connectivityLambda2Y{p in 1..np + 1, q in 1..np+1, i in 1..N
chiy[d, l] * (pos[p,i, 2] - pos[q,i,2] ) - Rcon >= epsilon + (-MConDim
- epsilon) * lambdaCon[p, q, i, d, l, 2];


subject to connectivityLambda1Z{p in 1..np + 1, q in 1..np+1, i in 1..N
chiz[d, l] * (pos[p,i, 3] - pos[q,i,3] ) - Rcon <= MConDim * (1 - lambda

subject to connectivityLambda2Z{p in 1..np + 1, q in 1..np+1, i in 1..N
chiz[d, l] * (pos[p,i, 3] - pos[q,i,3] ) - Rcon >= epsilon + (-MConDim
- epsilon) * lambdaCon[p, q, i, d, l, 3];
```

```
subject to testLambda{p in 1..np + 1, q in 1..np+1, i in 1..N}:
numberOf_lambdaCon[p,q,i] = sum{d in 1..Dvel, l in 1..Dvel/2, r in Dim}(la

subject to connectivity1{p in 1..np+1, q in 1..np+1, i in 1..N}:
(3*Dvel*(Dvel/2)) - sum{d in 1..Dvel, l in 1..Dvel/2, r in Dim}(lambdaCon

subject to connectivity2{p in 1..np+1, q in 1..np+1, i in 1..N}:
(3*Dvel*(Dvel/2)) - sum{d in 1..Dvel, l in 1..Dvel/2, r in Dim}(lambdaCon
epsilon + (- (3*Dvel*(Dvel/2)) - epsilon ) * bCon[p,q,i];



#Dataflow constraints

subject to dfAboveZero1{p in 1..np+1, i in 1..N, s in 1..np, j in 1..N }:
m[p,i,s,j] >= 0;

subject to dfAboveZero2{p in 1..np+1, q in 1..np+1, i in 1..N, s in 1..np
c[p, q, i,s,j] >= 0;


subject to dfAboveZero3{p in 1..np+1, j in 1..N, i in 1..j, s in 1..np }:
m[p,i,s,j] = 0;

subject to dfAboveZero4{p in 1..np+1, q in 1..np+1, j in 1..N, i in 1..j,
c[p, q, i,s,j] = 0;

subject to transmission1{s in 1..np, j in 1..N}:
m[s,j,s,j] = deltat * (cSensor  * bsensor[s,j] - sum{q in 1..np+1: q!=s}(

subject to transmission2{p in 1..np, j in 1..N, i in j+1..N, s in 1..np}:
m[p,i,s,j] = m[p,i-1,s,j] + deltat * ( sum{q in 1..np+1: q!=p}( c[q,p,i,s

subject to bufferSize{p in 1..np, i in 1..N}:
sum{s in 1..np, j in 1..i} (m[p,i,s,j]) <= hBar;
```

```
subject to timeNotGathered{p in 1..np, s in 1..np, j in 1..N, i in j..N
m[p,i,s,j] <= bsensor[p,j] * deltat * cSensor;

subject to passiveBase{s in 1..np, q in 1..np, j in 1..N, i in j..N}:
c[np+1, q,i,s,j] <= 0;


subject to connectiviteTransfer{p in 1..np, s in 1..np, q in 1..np+1, j
c[p,q,i,s,j] <= Cmax * bCon[p,q,i];

subject to collectiveOut{p in 1..np, i in 1..N}:
sum{ q in 1..np+1, s in 1..np, j in 1..i :p!=q }(c[p,q,i,s,j]) <= CmaxC

subject to collectiveIn{p in 1..np, i in 1..N}:
sum{ q in 1..np+1, s in 1..np, j in 1..i :p!=q }(c[q,p,i,s,j]) <= CmaxI


#Resouce constraints

subject to resource1:
sum{i in 1..N}(V[1, i]) * deltat <= battery1;
subject to resource2:
sum{i in 1..N}(V[2, i]) * deltat <= battery1;
```