NTNU – Trondheim
Norwegian University of
Science and Technology

Model Predictive Control in UAV Trajectory Planning and Gimbal Control

Bård Bakken Stovner

Master of Science in Cybernetics and Robotics
Submission date: June 2014
Supervisor: Tor Arne Johansen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

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

MSC THESIS DESCRIPTION SHEET

Name:	Bård Bakken Stovner
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Bruk av modellprediktiv kontroll i baneplanlegging og gimbalkontroll for UAVer (unmanned aerial vehicles)
Thesis title (English):	Model predictive control in UAV (unmanned aerial vehicle) trajectory planning and gimbal control

Thesis Description: Design and implement model predictive control methods for control of UAVs with pan-tilt gimbals.

The following items should be considered:

1. Designing and implementating the hardware in collaboration with other M.Sc. Candidates.
2. Describing the overall hardware and software infrastructure.
3. Modelling the behaviour of the UAV and gimbal system.
4. Implementing controllers for UAV and gimbal control.
5. Implementing the controllers in the designed software infrastructure.
6. Simulating and hardware-in-the-loop testing the controllers.
7. Conclude findings in a report.

Start date:	2014-01-20
Due date:	2014-06-16

Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Professor Tor Arne Johansen, Dept. of Eng. Cybernetics, NTNU
Co-supervisor:	MSc Frederik Stendahl Leira, Dept. of Eng. Cybernetics, NTNU

Abstract

In order to keep a target on the ground in the camera frame of a camera mounted with a pan-tilt gimbal to an unmanned aerial vehicle, three controllers that utilize the model predictive control methodology are developed. Through the process of simulating and hardware-in-the-loop testing the controllers, it is shown that one of the controllers can achieve satisfying control of both camera and vehicle. A new controller is proposed for future research which is assumed to achieve better control based on the findings of this thesis.

Keywords: search and rescue, unmanned aerial vehicle, trajectory planning, gimbal control, non-linear model predictive control, coordinate frame transformations, object tracking

Sammendrag

Målet for denne oppgaven er å finne en god metode for styring av ubemannete fly i søk og redningsoppdrag. Det blir foreslått tre modellprediktive kontrollere som har som mål å holde flyet nær et objekt på bakken samtidig som et kamera skal holde objektet i bilderammen. Flyet er utstyrt med en gimbal med frihetsgrader i pan og tilt, hvorpå kameraet er montert. En av kontrollerene viser seg gjennom simulering og hardware-in-the-loop-testing (HIL-testing) å kunne gi god styring av både fly og gimbal, slik at objektet forblir i bilderammen i hele forfølgelsesperioden. På bakgrunn av erfaringene med å simulere og HIL-teste de tre kontrollerene, blir er ny kontroller foreslått som antas å kunne forbedre styringen av fly og gimbal.

Problem Text

The main objective of this MSc thesis is to develop an autonomous unmanned aerial vehicle (UAV) flight control system for search-and-rescue (SAR) missions. A regular and an infrared (IR) camera are mounted with a pan-tilt gimbal (PTG) on the bottom side of the UAV for image gathering. By making use of an existing autopilot, i.e. Cloud Cap's Piccolo, the remaining tasks are to:

- Develop a trajectory planner, using non-linear model predictive control implemented in ACADO[1], which outputs trajectories for the autopilot and the PTG to follow
- Design and implement hardware payload on board the UAV
- Implement software for integrating the trajectory planner into an existing DUNE[2] architecture.

The reason for using ACADO is that it is a user-friendly toolkit which allows for simple and intuitive implementation of optimal control problems. An evaluation of ACADO and its suitability for the purpose of this thesis is therefore also in place.

[1] ACADO is a toolkit for optimal control and dynamic optimization.
[2] A runtime environment that allows for modularizing of the system and handles communication between modules.

Contents

List of Figures

List of Acronyms

AMOS Autonomous Marine Operations and Systems

AUV Autonomous Underwater Vehicle

BODY Body-fixed (frame)

CAM Camera-fixed (frame)

CGP Camera Ground Point

CMx Control Method x

CV Computer Vision

DH Denavit-Hartenberg

DOF Degree Of Freedom

ECEF Earth-Centered Earth-Fixed

FOV Field Of View

GUI Graphical User Interface

HIL Hardware-In-the-Loop

IR Infrared

LOS Line Of Sight

LSTS Laboratório de Sistemas e Tecnologias Subaquáticas

MCC MPC Command Center

MILP Mixed-Integer Linear Programming

MPC Model Predictive Control

NED North-East-Down

NMPC Non-linear Model Predictive Control

PCC Piccolo Command Center

PoE Power over Ethernet

PTG Pan-Tilt Gimbal

PWM Pulse Width Modulated

SAR Search-and-Rescue

UAV Unmanned Aerial Vehicle

1. Introduction

This thesis is written in cooperation with the Center for Autonomous Marine Operations and Systems (AMOS) which aims to meet the challenges related to safer maritime transport and monitoring and surveillance of coast and oceans.

1.1. Background and Motivation

Search and rescue (SAR) missions are missions aiming to locate and rescue missing persons. The search is often carried out by ground personnel, sometimes assisted by helicopters. Helicopters are, however, expensive both in procurement and day-to-day operation. Additionally, a lot of the functionality the helicopter provides, such as from-air rescue and fast transportation to hospital, may not always be needed or possible to use because of demanding terrain or other circumstances. In these cases, only the searching capabilities are utilized which might equally well be carried out by cheaper and smaller unmanned aerial vehicles (UAVs).

The US Department of Defence defined in 2005 a UAV as:

> *A powered, aerial vehicle that does not carry a human operator, uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or non-lethal payload. Ballistic or semi-ballistic vehicles, cruise missiles, and artillery projectiles are not considered unmanned aerial vehicles.* [U.S. Department of Defence, 2005]

It is commonly known that UAV technology is heavily researched on and utilized by the military of many countries in the world. With the rapid development of low-cost sensors and computer hardware in recent years, a market for commercial and private UAV applications has emerged. Search and rescue is one such application.

As the definition states, a UAV can be autonomous, which demands intelligent control design. This reduces the cost of man hours. On the other hand, with the use of a cheap communication link, a remote operator can be giving commands and replace some degree of autonomy, reducing the need for intelligent autonomous control implementation.

The relatively low cost also allows for using multiple UAVs to cooperate, performing a task much more efficiently than a single UAV or helicopter can. This may especially prove useful in SAR missions in which the search region is vast. However, lower cost is not the only advantage of UAVs in SAR missions. As the safety of a pilot is no longer a concern, the set of tasks that a UAV can perform is much larger than that of a manned aerial vehicle. Thus, UAVs can assist in nearly any scenario, whether it be natural disasters or extreme weather conditions.

In order to detect missing persons, the UAVs must be equipped with a camera. Typically, infrared (IR) cameras are used instead of regular cameras because separating a human being from its surroundings by thermal radiation is often a lot easier than by regular light. Good movement control of the IR camera will greatly enhance the searching efficiency. This can be done by a pan-tilt gimbal (PTG) with servomotors allowing panning and tilting motion.

The problem that this thesis concerns itself with is tracking an object on the ground by keeping it in the camera frame. This is important in order to get quality video footage of the object which ground personnel or a computer vision module can analyze. When too far away from an object, information about the object from the video can not be gathered. Therefore, this thesis will focus on the behaviour when in proximity of the object.

As the UAV and PTG used in this thesis has lower level controllers which follow setpoints, a control algorithm that generates such setpoints is needed. The control might be decoupled, i.e. controlling the UAV and PTG independently, or coupled, i.e. controlling the UAV in order to track the object more easily with the PTG and counteracting UAV movements with the PTG proactively. Coupled control might be better at keeping the object within the camera frame in the presence of disturbances as the PTG counteracts the movements of the UAV. However, it requires more computational power which is limited resource on the UAV. If coupled control is running too slowly on board the UAV, then using a decoupled control that is much faster might improve performance. As the need for a SAR mission may occur in demanding weather conditions, the control algorithm must also be able to handle weather phenomena, primarily wind.

Non-linear model predictive control (MPC) is a control algorithm based on a non-linear model, which is the best rendering of the dynamics of the UAV. It can plan a path taking wind into account, and find control actions on predicted future states, which might be useful for camera control. MPC takes a model of a system and a function expressing undesirable behaviour, and minimizes the function with respect to the model. Additionally, one can give the MPC boundaries which the behaviour of the system must stay within. This makes the MPC method robust, as the behaviour might be forced within safe limits, and versatile, as one has three inputs to define the system behaviour.

1.2. Previous Work

There exist several path planning and control methods for UAVs utilizing MPC. The versatility of the MPC methodology has resulted in very different approaches. Here, some of them along with existing gimbal control solutions on board UAVs will be presented.

Bellingham et al. [2002] has presented a trajectory optimization method for fixed-wing UAVs for reaching a target point flying through a cluttered environment. Mixed-integer linear programming (MILP) is used to formulate an optimization problem. The collision avoidance is included by adding constraints so that no part of the planned path collides with the rectangular approximated obstacles. This MILP formulation is used in the MPC framework, using a simple UAV kinematics model and the constraints mentioned. A short prediction horizon is used for the MPC in comparison with the time it takes to get to the target. This was compensated by a terminal cost consisting of an estimate of the time it takes for the UAV to reach the goal from the state at the end of the prediction horizon, which is found by a much simpler Dijkstra algorithm. This implementation utilizes the feedback property of the MPC method, in addition to addressing the time complexity issue of the MPC method by simplifying the problem past the prediction horizon.

[Yang and Sukkarieh, 2009] uses an adaptive non-linear MPC for tracking an already planned path through a cluttered environment. The existing path often consists of straight line segments which of course is impossible to follow exactly due to the path corners, i.e. where the line segments meet. The prediction horizon is shown to have a dampening effect on the trajectory, i.e. having a long prediction horizon cuts the corners of the path. Decreasing the prediction horizon, on the other hand, leads to a tighter tracking and sharper turns near the path corners. The length of the prediction horizon

is chosen by an adaptive law. This example shows another creative use of the MPC methodology, displaying nicely how the MPC formulation can be used to solve different types of problems.

The attempts to find gimbal controllers utilizing MPC were futile. However, some interesting aspects of gimbal control were found. Sun et al. [2008] presented a control algorithm for autonomous control of a UAV such that it tracks a moving target while using a fuzzy controller in order to point the camera towards a target on the ground. Rysdyk [2006] presents a gimbal controller aligning the normalized line-of-sight (LOS) vector of the camera to the unit vector pointing from the UAV towards the target. The UAV attempts to circle the object using a simple controller, while the gimbal controller is responsible for keeping the target in the field of view (FOV). Theodorakopoulos and Lacroix [2006] reviewed the gimbal types pan-tilt, tilt, and fixed. The pan-tilt gimbal was obviously found to be better if a proper gimbal controller was available. Sometimes, due to e.g. cost, weight, or size limitations, only tilt or fixed gimbals are available. In order for the camera to capture the interesting area, the flight path must enable it to. This raises an interesting point: that the UAV flight path can be found in order to enable and facilitate the gimbal to point stably at the target on the ground.

1.3. Contribution

The contribution to the field of UAV research by this thesis is exploring the possibility of on board model predictive control of unmanned aerial vehicles and pan-tilt controlled cameras. Model predictive control of the UAV will be compared to model predictive control of both UAV and gimbal simultaneously.

1.4. Outline

In Chapter 2, the background theory needed to understand the work done in this thesis is given. In Chapter 3 the hardware and software used in this thesis is described. A lot of the work that lies behind this chapter, mostly the hardware implementation, is done in collaboration with the other master thesis candidates Espen Skjong, Stian Nundal, and Carl Magnus Mathisen, and our PhD candidate supervisor Frederik Stendahl Leira.

In chapter 4 the kinematics of the UAV and gimbal is modelled, in order to be used in chapter 5 where the MPC formulations are stated and explained. In chapter 6 the software implementation of DUNE and ACADO is given. Finally, the results are stated in chapter 7 and 8, and discussed in chapter 9.

2. Background Theory

2.1. Robot Manipulator Representation

In order to describe the kinematics of the UAV and gimbal system used in this thesis, a common robot manipulator representation has been adopted. In this analysis, each degree of freedom is modelled as a *joint*. Joints may either be *revolute*, allowing rotational motion about an axis, or *prismatic*, allowing translatory motion along an axis, as shown in Figure 2.1. The joints are connected by *links*, which are stiff and represent the physical connection between two joints. At the end of the chain of joints and links there is an *end effector*, whose behaviour we are often interested in describing with respect to all the joints. The *Denavit-Hartenberg* (DH) convention provides a systematic procedure for performing this analysis, and relies on *transformation matrices*. [Spong et al., 2006]

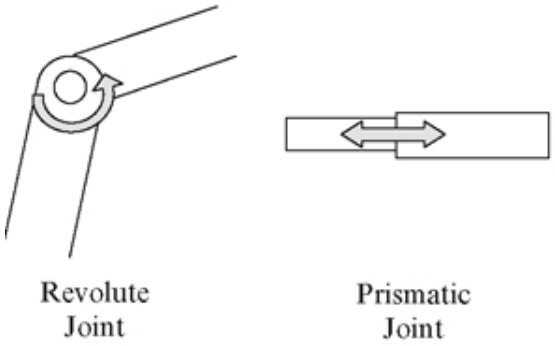


Figure 2.1.: Prismatic and revolute joints.[what-when-how.com, 2014]

2.1.1. Homogeneous Transformations

Homogeneous transformations are 4×4 matrix representations of rigid motions on the form

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{2.1}$$

in which $\mathbf{R} \in \mathbb{R}^{3\times 3}$ is a rotation matrix and $\mathbf{d} \in \mathbb{R}^3$ is a vector describing the distance between two points.

A *basic transformation* is either a rotation about an axis or translation along an axis, and is on the form

$$\bar{\mathbf{R}}_{w,\theta} = \begin{bmatrix} \mathbf{R}(w,\gamma) & \mathbf{0}_{3\times 1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{2.2}$$

or

$$\bar{\mathbf{T}}_{w,d} = \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{d}_w \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{2.3}$$

where $w \in [x, y, z]$ is the axis of motion, γ is a rotation about axis w, and $\mathbf{d}_w$ is the translation d along axis w. As will be seen in Section 2.1.2, only rotations about the x-axis and z-axis are needed. These are represented by the 3×3 rotation matrices:

$$\mathbf{R}(x,\delta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\delta & -s\delta \\ 0 & s\delta & c\delta \end{bmatrix} \tag{2.4}$$

$$\mathbf{R}(z,\gamma) = \begin{bmatrix} c\gamma & -s\gamma & 0 \\ s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

2.1.2. Denavit-Hartenberg Convention

In the Denavit-Hartenberg (DH) convention, each homogeneous transformation $\mathbf{A}_i$ is represented by a product of four basic transformations

$$\mathbf{A}_i = \bar{\mathbf{R}}_{z,\gamma_i}\bar{\mathbf{T}}_{z,d_i}\bar{\mathbf{T}}_{x,a_i}\bar{\mathbf{R}}_{x,\delta_i} \tag{2.6}$$

Let the robot model consist of n joints which are connected by $n-1$ links. The center of each joint q_i is denoted o_i. To each joint there is assigned a coordinate frame which has its origin in o_i and is fixed with respect γ_i, d_i, a_i, and δ_i. The z_i axis is required to point parallel to the axis of rotation or translation. Thus, the physical rotation or translation of a joint will always happen about or along the z_i axis. A rotation about the x_i axis may be needed in order to make the z_{i+1} axis point in the right direction.

The transformation matrix $\mathbf{A}_i$ represents the transition from coordinate frame $i-1$ to i, and we introduce the commonly used notation

$$\mathbf{T}_i^{i-1} = \mathbf{A}_i \tag{2.7}$$

The relationship between the initial coordinate frame and the end effector can now be found by:

$$\mathbf{T}_n^0 = \mathbf{T}_1^0\mathbf{T}_2^1\cdots\mathbf{T}_n^{n-1} = \mathbf{A}_1\cdots\mathbf{A}_n \tag{2.8}$$

A note will be made on the general use of this notation. Let the vector $\boldsymbol{v}^B$ be described in the coordinate system B. Using the notation, in order to find $\boldsymbol{v}$ described in coordinate system A, $\boldsymbol{v}^A$, the vector needs to be multiplied with the transformation matrix $\boldsymbol{T}_B^A$, i.e.

$$\boldsymbol{v}^A = \boldsymbol{T}_B^A\boldsymbol{v}^B \tag{2.9}$$

If $\boldsymbol{T}_B^A$ and $\boldsymbol{v}^A$ are known, $\boldsymbol{v}$ can easily be found by using the transformation matrix property

$$\boldsymbol{T}_A^B = \begin{bmatrix} \boldsymbol{R}_B^{A\top} & -\boldsymbol{R}_B^{A\top}d \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{2.10}$$

[Spong et al., 2006].

2.2. Coordinate Frames

In navigation, transitions between coordinate frames are essential for breaking down the complex relationships between global and local position and orientation to smaller steps which are intuitive and manageable for humans. Here, a short review of the coordinate frames relevant to this thesis is given.

2.2.1. Geodetic Coordinates

Geodetic coordinates are given by the three parameters longitude, l; latitude, μ; and height, h. The earth is approximated as an ellipsoid with parameters r_e and r_p, called the Earth's equatorial and polar radii, respectively, which are the semi-axes of the ellipsoid. This ellipsoid approximation can be seen in Figure 2.2, and we recognize the longitude and latitude parameters. The height, h, is not shown in the figure, but is the height above the ellipsoid, i.e. a length along the z_n axis. [Fossen, 2011]

2.2.2. Earth-Centered Earth-Fixed (ECEF) Frame

The Earth-centered Earth-fixed reference frame $\{e\} = (x_e, y_e, z_e)$ has its origin, o_e, in the center of the earth, and rotates with the earth. Its z-axis points along the earth's axis of rotation upwards, i.e. towards the geographic north pole. The x and y axes point to equator, with x pointing towards Greenwich, which is defined as zero longitude. Equator is defined as zero latitude. [Fossen, 2011]

2.2.3. North-East-Down (NED) Frame

The North-East-Down (NED) coordinate system $\{n\} = (x_n, y_n, z_n)$, with origin o_n, is defined relative to the Earth's reference ellipsoid (World Geodetic System, 1984). Its x and y axes lie in the local tangent plane of the earth, with the x-axis pointing north and the y-axis pointing east. The z-axis points perpendicularly to the tangent plane downwards. The location of $\{n\}$ relative to $\{e\}$ is determined using the longitude and latitude, as can be seen in Figure 2.2. The position and attitude of a vehicle are often described in the NED frame. [Fossen, 2011]

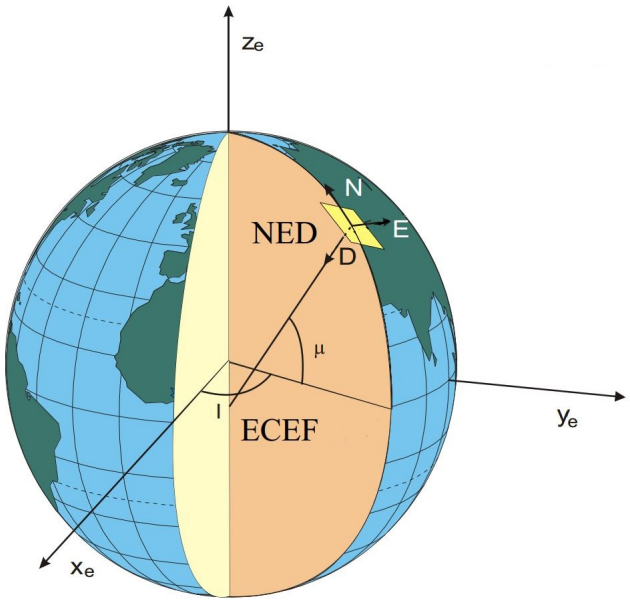


Figure 2.2.: ECEF and NED frames. [Fossen, 2014]

2.2.4. Body-Fixed (BODY) Frame

Figure 2.3 shows the body-fixed (BODY) reference frame $\{b\} = (x_b, y_b, z_b)$, with origin o_b, which is a moving coordinate frame that is fixed to the vehicle. The position and attitude of the vehicle is described in the NED or ECEF frame, while linear and angular velocities are described relative to the BODY frame. Its x-axis points along the centerline of the vehicle towards the nose, while its z-axis points downwards through the bottom of the vehicle. The y-axis is assigned such that the coordinate system is a right-hand frame. [Fossen, 2011]

2.2.5. Camera-Fixed (CAM) Frame

The camera-fixed (CAM) reference frame $\{c\} = (x_c, y_c, z_c)$, with origin o_c, is a moving coordinate system that is fixed to the camera mounted on the vehicle. Its z-axis points from the camera lens and outwards, while its x and y axes point up and right relative to the image, respectively.

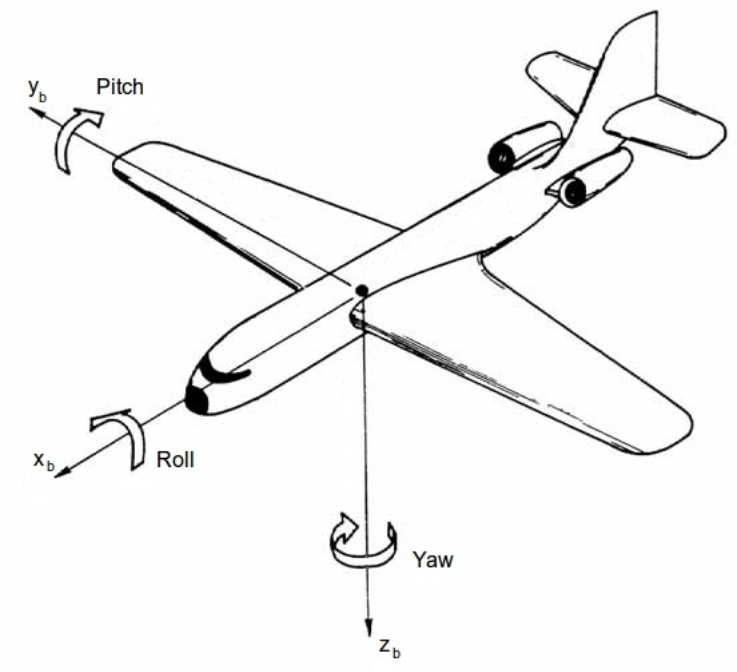


Figure 2.3.: BODY frame. [Mahoney, 2014]

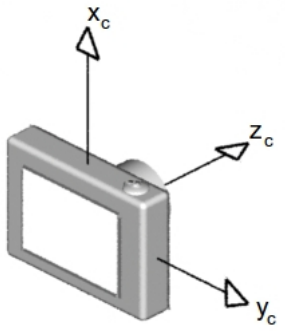


Figure 2.4.: CAM frame.

2.3. Model Predictive Control

State feedback non-linear model predictive control (NMPC) is a control algorithm that attempts to minimize a cost function with respect to the modelled system dynamics and constraints on the state and control variables. When the optimal control variables for the system over a time horizon is found, only the first time step of the control variables is used as inputs to the system. When the time step is over, new control variables are found with updated measurements or estimates of the states.

Let $\boldsymbol{x}$ be a vector of states and $\boldsymbol{u}$ a vector of control inputs. The mathematical model of the system that is to be controlled is denoted $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u})$. Additionally, one is allowed to specify *path constraints*, $\boldsymbol{s}(\boldsymbol{x}, \boldsymbol{u})$ which limit the behaviour of the system. Now, the goal of NMPC algorithm is to minimizes a *cost function*, $\Phi(t) = \|\boldsymbol{h}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) - \boldsymbol{\eta}\|^2_{\boldsymbol{Q}}$, over a time interval of length T called the *prediction horizon*. $\boldsymbol{h}(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$ is a function expressing what is to be controlled towards a time-varying set points $\boldsymbol{\eta}(t)$ and $\boldsymbol{Q}$ is a positive definite diagonal matrix. This NMPC formulation can be written

$$\min_{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)} \int_{t_0}^{t_0+T} \|\boldsymbol{h}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) - \boldsymbol{\eta}(t)\|^2_{\boldsymbol{Q}} dt + \|\boldsymbol{m}(\boldsymbol{x}(t_0+T), t_0+T) - \boldsymbol{\mu}\|^2_{\boldsymbol{R}} \tag{2.11a}$$

s.t.

$$0 = \boldsymbol{f}(t, \boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \boldsymbol{u}(t)) \tag{2.11b}$$

$$0 \geq \boldsymbol{s}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2.11c}$$

$$0 = \boldsymbol{r}(\boldsymbol{x}(t_0+T), (t_0+T) \tag{2.11d}$$

The second term in equation (2.11a) expresses the terminal cost, which may be included in order to drive the system towards a desired state at the end of the horizon. Equation (2.11d) may be included in order to force the system to a desired state at the end of the horizon. These two terms are mentioned here as general information, and are not used in this thesis. [Houska et al., 2009–2013]

A note about the prediction horizon T will be made. The prediction horizon is the length of time that the behaviour of the system is predicted and attempted optimized for. That means that anything that may happen beyond the prediction horizon will not be taken into account when finding an optimal control input. Therefore, in order to get

good results, it is important to set the prediction horizon long enough that the system converges or all the interesting behaviour is taken into account.

2.3.1. Slack Variables in MPC Formulation

Equation (2.11c) is in this thesis used to place box constraints on some of the states and control inputs. This means that the specified variables must be contained within certain brackets. If these intervals are too small, the MPC algorithm may not be able to find a solution. Another problem that may be encountered is that the algorithm may, mainly due to disturbances, find itself outside the feasible region, i.e. outside the space where all constraints are satisfied, with no way of returning to the feasible region. A solution to these two problems is to augment the current problem with *slack variables*. Slack variables allow the MPC algorithm to break certain constraints, guaranteeing that the constraint will not keep the algorithm from finding a solution. Since slack variables expand the feasible region, one may not find oneself outside it.

In addition to adding slack variables to the box constraints, the slack variables must be minimized heavily in the cost function. One way of adding slack variables to the MPC formulation is to expand the control variables vector u with the desired number of slack variables and include the extra control variables to the cost function and box constraints. Using an arbitrary example model with the variables, constraints, and cost function

$$
\begin{aligned}
\min_{\boldsymbol{x}(\cdot),u(\cdot)} \int_{t_0}^{t_0+T} \|\boldsymbol{h}(t,\boldsymbol{x}(t),u(t)) - \boldsymbol{\eta}(t)\|^2_{\boldsymbol{Q}} dt \\
\boldsymbol{x} = [x_1, x_2]^\top \\
u = u_1 \\
\boldsymbol{h}(t,\boldsymbol{x}(t),u(t)) = \begin{bmatrix} h_1(t,\boldsymbol{x}(t),u(t)) \\ h_2(t,\boldsymbol{x}(t),u(t)) \end{bmatrix} \\
\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},u) \\
x_{1min} \leq x_1 \leq x_{1max} \\
x_{2min} \leq x_2 \leq x_{2max},
\end{aligned}
$$

slack variables variables may added on x_1 and x_2:

$$\begin{aligned}
\min_{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)} \int_{t_0}^{t_0+T} & \|\boldsymbol{h}(t,\boldsymbol{x}(t),\boldsymbol{u}(t)) - \boldsymbol{\eta}(t)\|^2_{\boldsymbol{Q}} dt \\
\boldsymbol{x} &= [x_1, x_2]^\top \\
\boldsymbol{u} &= [u_1, s_1, s_2]^\top \\
\boldsymbol{h}(t,\boldsymbol{x}(t),\boldsymbol{u}(t)) &= \begin{bmatrix} h_1(t,\boldsymbol{x}(t),u_1(t)) \\ h_2(t,\boldsymbol{x}(t),u_1(t)) \\ s_1 \\ s_2 \end{bmatrix} \\
\dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u}) \\
x_{1min} \leq x_1 + s_1 &\leq x_{1max} \\
x_{2min} \leq x_2 + s_2 &\leq x_{2max},
\end{aligned}$$

Additionally, the weight matrix $\boldsymbol{Q}$ is augmented from $\boldsymbol{Q} = diag([q_1, q_2])$ to $\boldsymbol{Q} = diag([q_1, q_2, q_3, q_4])$. It is important that q_3 and q_4 are sufficiently large relative to q_1 and q_2 such that the slack variables eventually are forced to zero. This means that the system with slack variables should converge to the same solution as the system without slack variables would if there were no infeasibility issues.

2.4. ACADO

ACADO Toolkit is a user-friendly toolkit written in C++ for automatic control and dynamic optimization. ACADO is of interest for this thesis because it offers a general framework for formulating and solving optimization problems including NMPC problems. One of the key functions of ACADO is the code export functionality, which generates a library with C functions allowing much faster calculations.[Houska et al., 2009–2013]

2.4.1. Gauss Newton Hessian Approximation

ACADO uses a sequential quadratic programming (SQP) algorithm in an attempt to find an optimal step $\boldsymbol{p}_k$ such that $\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \boldsymbol{p}_k$, $\boldsymbol{z} \in \mathbb{R}^n$. Starting with the least

squares problem

$$\min_{\boldsymbol{z}} \Phi(\boldsymbol{z}) = \|\boldsymbol{g}(\boldsymbol{z})\|_2^2 = \boldsymbol{g}(\boldsymbol{z})^\top \boldsymbol{g}(\boldsymbol{z}) \tag{2.12}$$

where $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^m$, ACADO uses the Newton method which finds $\boldsymbol{p}_k$ by

$$\boldsymbol{p}_k = -(\nabla^2 \Phi(\boldsymbol{z}))^{-1} \nabla \Phi(\boldsymbol{z})$$

By defining $\boldsymbol{J}(\boldsymbol{z}) = \boldsymbol{g}'(\boldsymbol{z})$ we can find

$$\nabla \Phi(\boldsymbol{z}) = \boldsymbol{J}(\boldsymbol{z})^\top \boldsymbol{g}(\boldsymbol{z})$$
$$\nabla^2 \Phi(\boldsymbol{z}) = \boldsymbol{J}(\boldsymbol{z})^\top \boldsymbol{J}(\boldsymbol{z}) + \sum_{i=1}^{m} \boldsymbol{f}_i(\boldsymbol{z}) \nabla^2 \boldsymbol{f}_i(\boldsymbol{z})$$

When exporting an ACADO MPC project to C-code, ACADO uses a commonly used Hessian approximation method called the Gauss Newton method. The Gauss Newton method assumes that

$$\sum_{i=1}^{m} \boldsymbol{f}_i(\boldsymbol{z}) \nabla^2 \boldsymbol{f}_i(\boldsymbol{z}) \approx 0 \tag{2.13}$$

which greatly simplifies the optimization as the second order derivative terms $\nabla^2 \boldsymbol{f}_i(\boldsymbol{z})$ are neglected. This simplification introduces an error which is small near a local minimum, but may be large far away. [Gratton et al., 2004]

2.5. DUNE: Unified Navigational Environment

DUNE is a runtime environment for software on board unmanned systems made by the Laboratório de Sistemas e Tecnologias Subaquáticas (LSTS) in Porto, Portugal. It provides an architecture and operating system independent platform that handles the communication between the different modules that collectively carries out the operation of the program. These modules may e.g. be navigation, control, and communication from and to sensors, actuators, and ground stations.

In DUNE, the user is allowed to partition a program into smaller user-defined *tasks* with built-in communication interfaces to a *message bus* shared between all the tasks. Tasks may share information by *dispatching* messages to the message bus and may

receive information by *consuming* messages from the message bus. In order for a task to consume a specific type of messages from the message bus, it must subscribe to that type of messages. These messages are defined by a protocol called IMC, which contains a lot of predefined massages. IMC also supports creation of new user-defined message types.

3. System Description

In this chapter the entire system is described. First, the control hierarchy of the system is described in order to show the place of the NMPC trajectory planner. Then, a coarse system overview is given. Lastly, a detailed description of the entire system covering hardware implementation and the communication between software modules will be given.

3.1. Control Hierarchy

The goal of this section is to show the place of the NMPC trajectory planner in the control system of the UAV. Figure 3.1 shows the structure of the control system. On top, there is a CV module that registers and keeps track of all objects. A list of these objects is then sorted by order of visitation by a object priority queuer, which also tells the NMPC trajectory planner which object to track. The NMPC module then sends set points to the Piccolo autopilot and gimbal controller. The autopilot and gimbal controller are then to track these set points. Now, the bottom layer in figure 3.1 consists of more complexity than is shown in the figure, but only the interface to the trajectory planner is of interest to this thesis.

3.2. System Architecture

In Figure 3.2, a coarse overview of the system is described. The system consists of

- Penguin B UAV - A small unmanned aerial vehicle shown in Figure 3.2. It has a wing-span of 3.3 meters and can handle up to 11.5 kg of fuel and payload[UAV Factory, 2014b]. On board it will carry:
  - Piccolo SL - Autopilot and telemetry

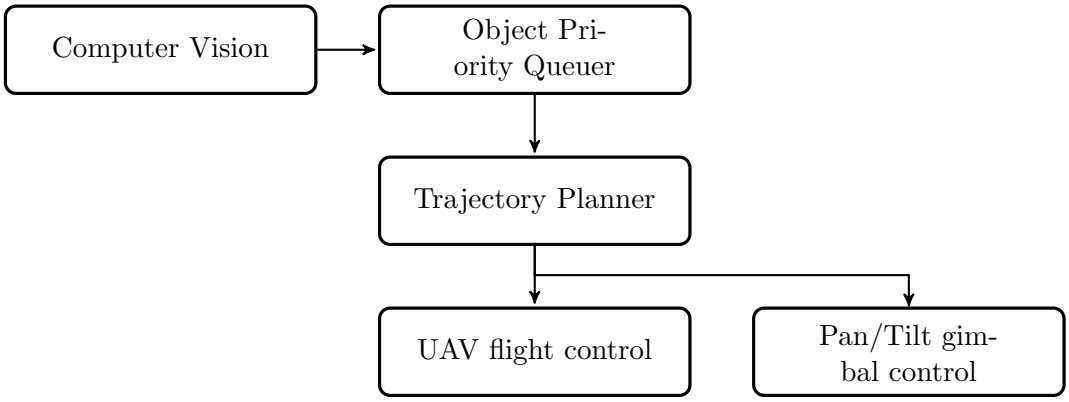


Figure 3.1.: Control system hierarchy

  - Designed payload - Contains communication with ground station and piccolo, and hardware for running MPC and CV modules

- Satellite link - GPS measurements
- Ground station computer with
  - MPC Command Center (MCC) software communicating with the payload over $5.6GHz$ link.
  - Piccolo Command Center (PCC) communicating with the Piccolo over a $2.4GHz$ link

3.3. Payload Hardware

3.3.1. Generator

On board is the 3W 28i engine with 80W generator system delivered by the UAV Factory, capable of delivering 80W and 12V[UAV Factory, 2014a].

3.3.2. PandaBoard

PandaBoard is a small low-cost single-board computer made by a joint effort of volunteer developers. It offers several connectors that is useful for the project, i.e. RS-232, Ethernet, and USB, in addition to an SD card slot. An SD card will hold a stripped down

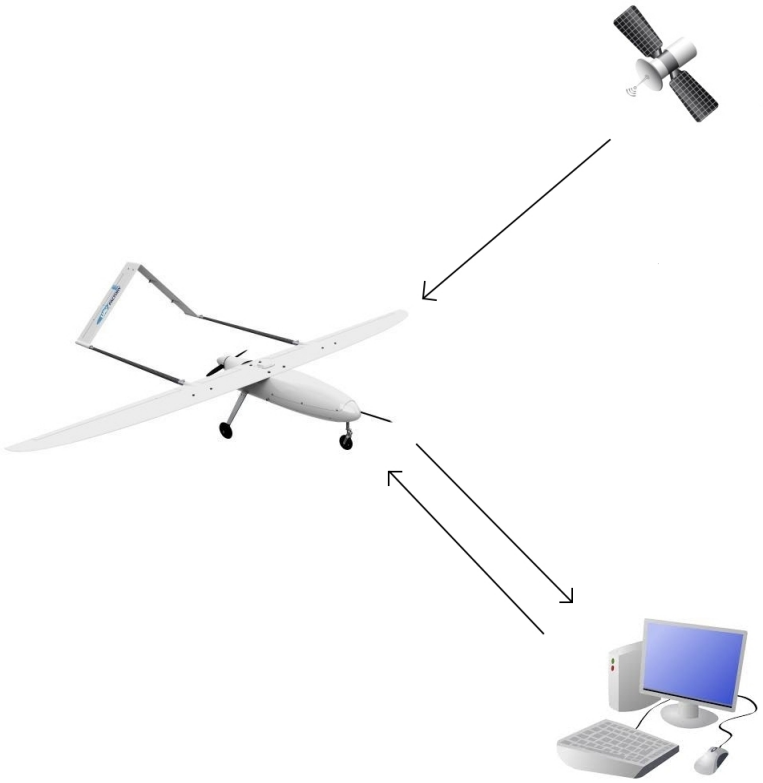


Figure 3.2.: Coarse description of system.

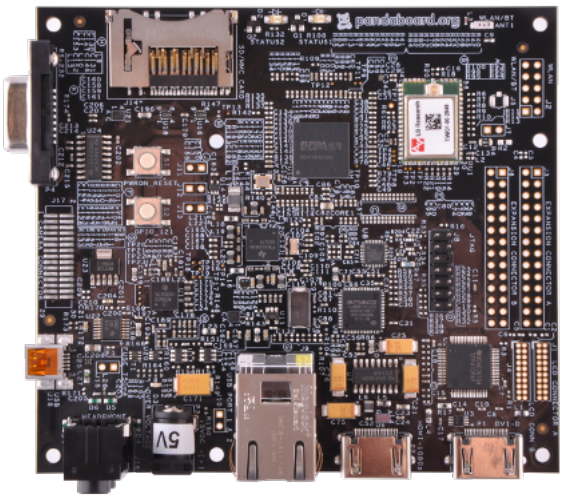


Figure 3.3.: PandaBoard.

Linux operating system distribution and the developed software to be run in the payload. The PandaBoard is powered through a power inlet taking $5V$. [pandaboard.org, 2014]

3.3.3. FLIR Tau 2 Thermal Imaging Camera

The FLIR Tau 2 thermal imaging camera is used to capture IR video stream. Its specifications are:

- Power supply - requires 4.4 - 6.0 V which can be supplied through mini USB
- Analog Video Channel - provides analog video signal via a coaxial cable with 75Ω characteristic impedance
- A focal length of 19mm, and horizontal and vertical angle of view of 32° and 26°, respectively

[FLIR, 2014]

Figure 3.4.: Axis Framegrabber.

3.3.4. Axis M7001

Axis M7001 is a analog-to-digital (AD) video encoder from Axis Communications which can be seen in Figure 3.4. It receives an analog video stream from a coaxial cable through the BNC connector shown on the right hand side of Figure 3.4. An RJ-45 Ethernet connector, which can be seen on the left hand side of Figure 3.4, outputs the digital video stream and supplies the Axis with a 48 V power signal.

- Power supply - requires power through Power over Ethernet (PoE) IEEE 802.3af Class 2, a standard that requires 37 - 57 V. 48 V is used in this project.
- Analog Video Input - The analog video input is received through a 75Ω coaxial video cable to the BNC connector shown on the right hand side of Figure 3.4
- Digital Video Output - The digital video output is sent out through an Ethernet cable

The PoE and the digital video are sent through the same Ethernet cable using a PoE Injector cable, which is shown in Figure 3.5. The 48 V power signal is connected to the circular connector in Figure 3.5, while the digital video stream can be gathered from the RJ-45 male cable. The female Ethernet connectors of the Axis and the PoE Injector are connected with a regular Ethernet cable. [Axis Communications, 2011]

3.3.5. BTC88 Standard Gimbal

The BTC88 Standard Gimbal is used in this project for controlling the IR camera. It is a pan-tilt gimbal which has the following specifications[Micro UAV, 2011]:

Figure 3.5.: PoE injector cable.

Figure 3.6.: BTC88 Standard Gimbal.

- Power supply - 12 V
- Pan Range - the pan angle is restricted between $-180°$ and $180°$
- Panning Speed - the gimbal can pan $360°$ in less than two seconds giving it a maximum speed of $\dot{\alpha}_{\max} = \frac{360°}{2s} = \frac{2\pi}{2}\frac{rad}{s} = \pi\frac{rad}{s}$
- Tilt Range - $10°$ to $90°$ degrees down, meaning it can point from $10°$ up from the vertical axis to parallel with the horizontal plane when mounted on a horizontal plane.
- Tilting Speed - the gimbal can tilt $90°$ in $\frac{1}{2}$ second giving it a maximum speed of $\dot{\beta}_{\max} = \frac{90°}{\frac{1}{2}s} = \frac{\frac{\pi}{4}}{\frac{1}{2}}\frac{rad}{s} = \frac{\pi}{2}\frac{rad}{s}$

[Micro UAV, 2011]

Figure 3.7.: Cloud Cap Technology's Piccolo autopilot.

3.3.6. Piccolo SL Autopilot

The Piccolo SL autopilot provides a complete integrated avionics solution that includes the flight control processor, inertial sensors, ported air data sensors, GPS receiver and datalink radio[Cloud Cap, 2014]. There is support in the Piccolo SL for pan-tilt gimbals, allowing the pan and tilt angle messages to travel through the Piccolo. It also supports hardware-in-the-loop (HIL) pre-flight testing. It requires a 12V power input.

3.3.7. Rocket M5

The Rocket M5 is a MIMO radio used to communicate with the ground stationed PC which will be monitoring the UAV and giving high level commands. The specifications of the Rocket M5 are [Ubiquiti Networks, 2014]:

- Power Supply - 24V through PoE
- Operation Frequency - 5470-5825 MHz

3.3.8. Step-Up Converter

As the Axis M7001 needs 37-57V to operate, a step-up converter is needed. A 12V to 48V step-up converter was purchased on Ebay for this project by PhD candidate Frederik Stendahl Leira, who is project leader for this project.

3.3.9. Step-Down Converter

As the PandaBoard, network switch, and the IR camera need 5V to operate, a step-down converter is needed. The step-down converter used in this project is a 12V to 5V Chuangruifa converter, which has a maximum load of 3A.

3.3.10. Connection

In this section, the physical connection between the hardware components will be described. In Figure 3.8 a diagram over the hardware components and their connectors can be seen. In figure E.1 the payload can be seen.

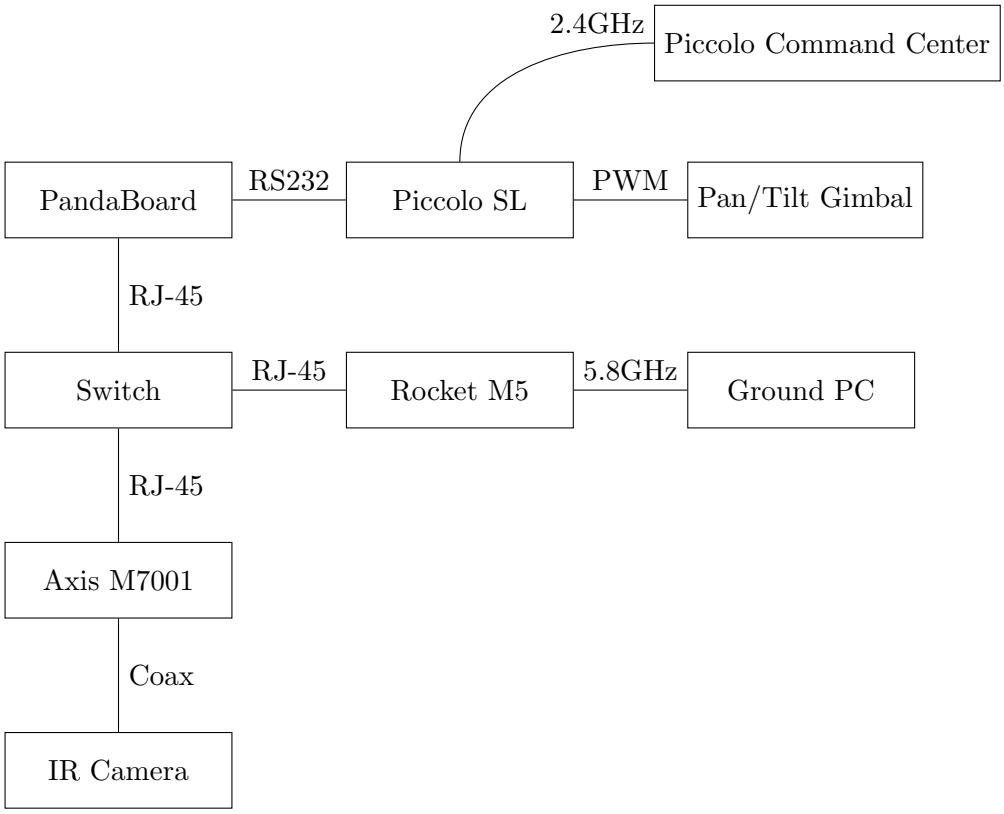


Figure 3.8.: Connection diagram.

The connections need to be described:

- RJ45 - A standard Ethernet cable

- RS232 - Serial link over a RS232 interface
- Coax - A coaxial cable with 75 Ω characteristic impedance
- PWM - A pulse-width modulated (PWM) power signal
- 2.4GHz - A 2.4GHz radio link
- 5.8GHz - A 5.8GHz radio link

3.3.11. Power Plan

In this section, a full overview over the power consumption of the different hardware components was supposed to be given. Unfortunately, as most of the components have lacking or no documentation, this attempt was futile. However, an overview of the power distribution will be given and the maximum payload power consumption will be stated.

The generator supplies the payload with 12V, which is used to power the Piccolo, the gimbal, and the step-up and step-down converters. The 48V signal from the step-up converter is used to power the Axis M7001, while the 5V signal from the step-down converter is used to power the PandaBoard, network switch, and IR camera. This can be seen in Figure 3.9.

The generator can, as mentioned, deliver maximum 80W. The power consumption of the navigation system is roughly 10W, and 10W should be left as a buffer. That leaves 60W to power the payload during operation.

3.4. Software

In this section, a description of the different software modules and how they interact with each other will be given. Only the parts of the modules that has a relevance to this thesis project is described.

Dune, which was described in Section 2.5, is the backbone of the software system. It handles the communication between all the tasks that together determines the behaviour of the program.

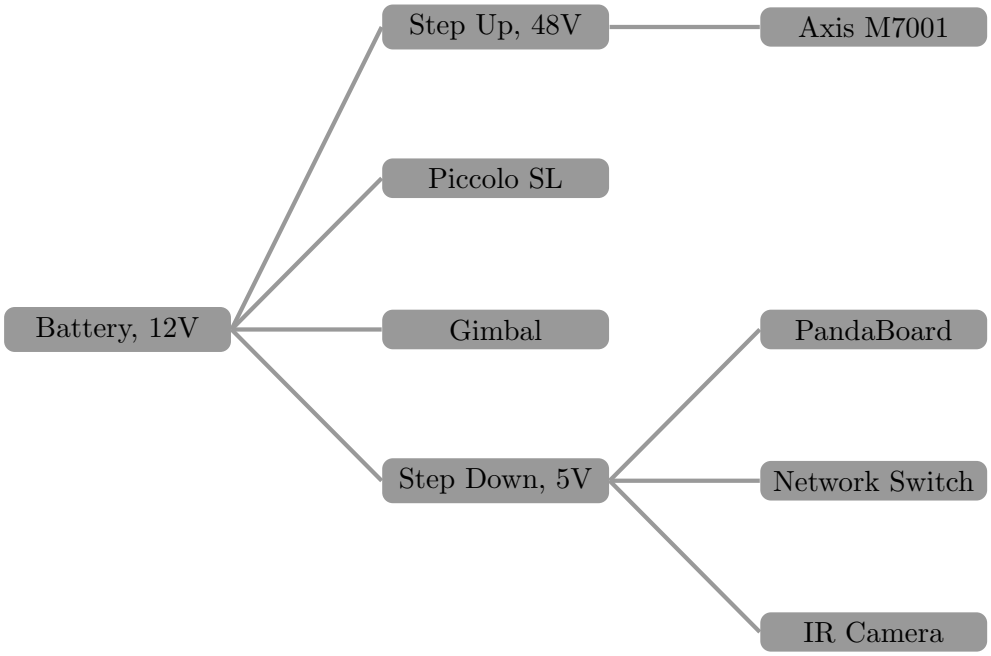


Figure 3.9.: Power distribution in the system.

3.4.1. MPC Command Center

Because there are several other master thesis projects working with UAV control, there must be a high level of control, determining which control module is to run. The MPC Command Center (MCC) is a program written by M.Sc. candidates Espen Skjong and Stian Nundal for their master thesis, which does this job. It is run from a ground stationed PC and gives an operator the control over which control module is allowed to run. The MCC sends an IMC message stating which control module is allowed to run to a communication module on board the UAV. This communication module is created so that the MCC only has one interface to the PandaBoard on board the UAV. The MCC expects an answer from the communication module that has been turned on or off, confirming that the message was received.

3.4.2. Communication Module

The communication module communicating with the MCC is a simple DUNE task, written by M.Sc. candidate Carl Magnus Mathisen. It consumes the message from the MCC that turns control modules on or off, and relays them to the correct control task.

It also listens for confirm messages from the control tasks and relays them to the MCC. This behaviour is described in Figure 3.10.

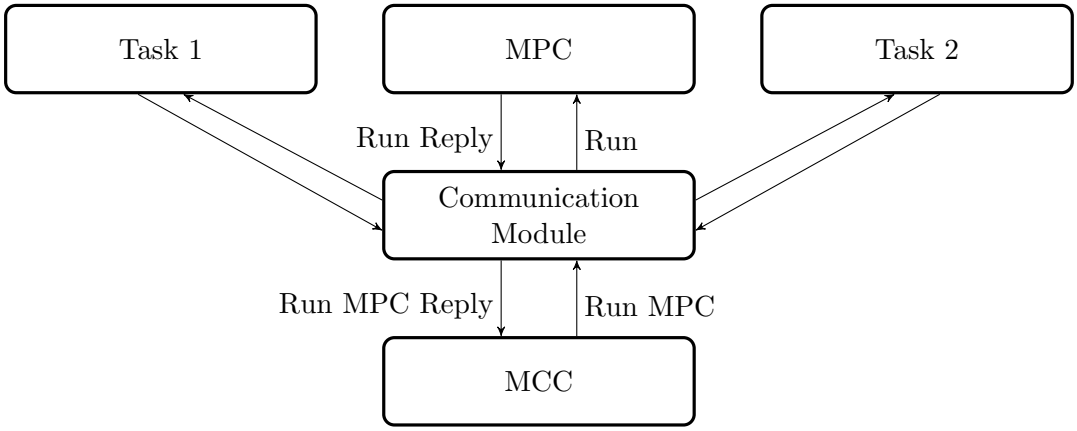


Figure 3.10.: Communication with MCC.

3.4.3. Piccolo Autopilot

The Piccolo autopilot has a DUNE interface that allows it to dispatch and consume IMC messages to/from the Dune message bus. From the Piccolo, UAV position and wind estimate messages are dispatched. The Piccolo has an interface to pan-tilt-zoom gimbals, which makes it possible to send the pan and tilt angle messages to the gimbal through the Piccolo. Thus, it receives messages containing roll, pan, and tilt angles that it attempts to track. Additionally, in order to control the UAV with a roll set point angle, the Piccolo must be told to allow it. This is depicted in Figure 3.11

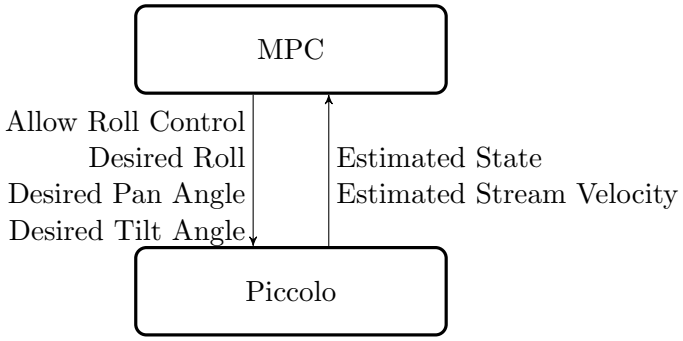


Figure 3.11.: Communication with Piccolo autopilot.

3.4.4. Controller Task

The controller task is the task that contains the ACADO MPC controller. It communicates with the communication module as described in Section 3.4.2 and the Piccolo as described in Section 3.4.3. In the case of CM1, the gimbal control is actually done in a separated task in order to calculate the gimbal angle set point more often. The functionality of these to tasks combined, however, are as described above.

4. Modelling

4.1. Assumptions and Simplifications

In order to simplify the mathematical modelling, some simplifications and assumptions are made. In this section, they will be presented.

4.1.1. Origin of CAM and BODY Frames Coincide

The origin of the CAM and BODY frames are approximated to coincide. This greatly simplifies the equations describing the relationship between UAV and gimbal angles and the camera attitude in the NED frame. Consequently, the model used in the MPC is simplified which improves runtime performances. This runtime improvement comes at the cost of the accuracy of the model, as the simplification will introduce an error. This error, however, is assumed to be negligible due to the small distance between the origin of the CAM and BODY frames compared to the distance from the UAV to the ground.

4.1.2. Local Flat Earth Approximation

The UAV's and the object's position is referenced in the NED frame, which assumes that the earth can be approximated as flat locally. When operating over sufficiently small areas, this is a good approximation. If the UAV travels far away from the origin of the NED frame, and the approximation leads to significant errors, the NED frame can be updated and moved to a more convenient position.

The benefit of using the NED frame in modelling of the UAV is huge. Assuming that the earth is flat locally means that the east and north position of the UAV can be updated by the simple equations derived in (4.13a)-(4.13b). The downside of this approximation is the errors it introduces. However, if the NED frame is updated often enough by finding

new frames as the UAV travels, this error will be small. Exactly this solution is already implemented in DUNE.

4.1.3. Constant height

The Piccolo autopilot is assumed to keep the UAV at a constant height over ground. This means that the height dynamics does not need to be modelled and the pitch angle can be assumed to be zero. Consequently, the pitch rate can also be assumed to be zero.

$$\dot{D} = 0 \tag{4.1}$$

$$\theta = \dot{\theta} = 0 \tag{4.2}$$

The effects of this simplification is that two degrees of freedom are removed from the model. This leaves us with a simpler model, which will make the MPC implementation computationally less heavy. The height will be included in the MPC as it is important to get good estimates of the vertical distance between the UAV and object, but it will be assumed constant in the prediction horizon.

4.1.4. Constant True Airspeed

The Piccolo autopilot is assumed to keep the UAV at a constant airspeed of $V = 28\frac{m}{s}$. The true airspeed is the speed of the UAV relative to the air surrounding it. The true air velocity decomposed in the BODY frame is $\boldsymbol{V}_t^b = [V, 0, 0]^\top$.

4.2. Transformation Between Frames

In Section 2.2, an overview of the frames relevant to this thesis was given. A robot manipulator description was explained, and here the transformations between the frames will be derived.

4.2.1. Geodetic Coordinates to ECEF

For the transformation from geodetic coordinates to ECEF, we first need to know the Earth's equatorial and polar radii

$$r_e = 6378137m$$
$$r_p = 6356752m$$

The transformation can be found in e.g. [Vik, 2012]:

$$\boldsymbol{p}^e = \begin{bmatrix} (N_r + h)\cos(\mu)\cos(l) \\ (N_r + h)\cos(\mu)\sin(l) \\ (\frac{r_p^2}{r_e^2}N_r + h)\sin(\mu) \end{bmatrix} \tag{4.3}$$

where N_r is obtained from

$$N_r = \frac{r_e^2}{\sqrt{r_e^2\cos^2(\mu) + r_p^2\sin^2(\mu)}}$$

4.2.2. From ECEF to NED

The transition from ECEF to NED can be found in [Fossen, 2011]:

$$\boldsymbol{R}_e^n = \begin{bmatrix} -\cos(l)\sin(\mu) & -\sin(l)\sin(\mu) & \cos(\mu) \\ -\sin(l) & \cos(l) & 0 \\ -\cos(l)\cos(\mu) & -\sin(l)\cos(\mu) & -\sin(\mu) \end{bmatrix} \tag{4.4}$$

When used in this thesis, the position of a NED reference frame that is fixed on the earth's surface will be known in LLH coordinates, and the position of the UAV with respect to the reference frame is what needs to be found. This is expressed as:

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix} = \boldsymbol{p}_{UAV}^n = \boldsymbol{R}_e^n(\boldsymbol{p}_{UAV}^e - \boldsymbol{p}_{ref}^e) \tag{4.5}$$

$\boldsymbol{p}_{ref}^e$ and $\boldsymbol{p}_{UAV}^e$ are found using (4.3) with inputs (l_{ref}, μ_{ref}) and $(l_{UAV}, \mu_{UAV}, h_{UAV})$, respectively.

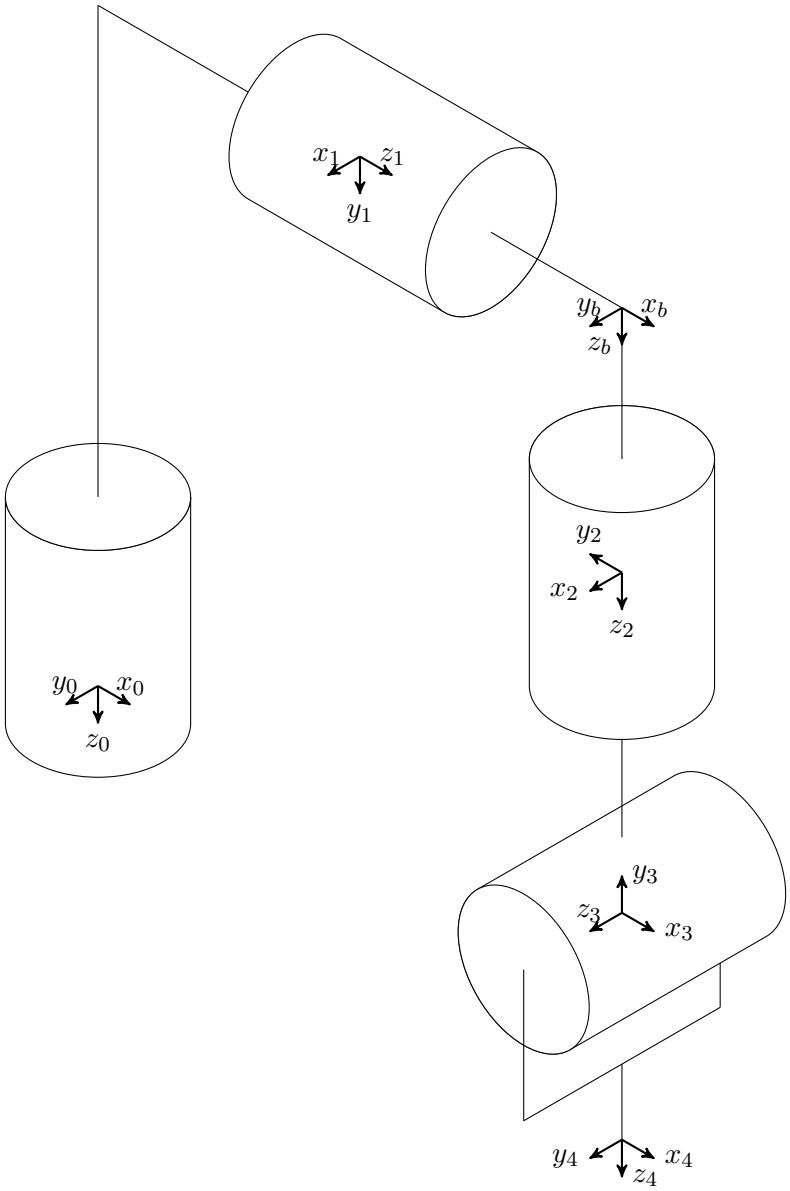


Figure 4.1.: Robot manipulator description of the UAV.

4.2.3. From NED to BODY

For the transformation from NED to BODY, the coordinate system 0 in Figure 4.1 needs to be explained. Its origin, o_0, is centred in the UAV coinciding with the origin of the BODY frame. However, it is aligned precisely like the NED reference frame. This means that x_0 points northwards from o_{ref}, y_0 points eastwards from o_{ref} and z_0 points downwards as described in Figure 2.2. This is the local flat earth approximation made in Section 4.1.

Using the DH convention of Section 2.1.2, the transformation from NED to BODY can be found by looking at Figure 4.1.

$$\boldsymbol{A}_1 = \boldsymbol{R}_{z,\psi+\frac{\pi}{2}}\boldsymbol{R}_{x,\frac{\pi}{2}} \tag{4.6}$$

$$\boldsymbol{A}_2 = \boldsymbol{R}_{z,\phi}\boldsymbol{R}_{x,-\frac{\pi}{2}} \tag{4.7}$$

From coordinate system 2, we see that a rotation of $-90\,^\circ$ about the z_2 axis needs to be performed in order to go from coordinate system 2 to BODY. This yields

$$\begin{aligned}\boldsymbol{R}_b^n &= \boldsymbol{R}_b^0 = \boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{R}_{z,-\frac{\pi}{2}} \\ \boldsymbol{R}_b^n &= \begin{bmatrix}\cos(\psi) & -\sin(\psi)\cos(\phi) & \sin(\psi)\sin(\phi) \\ \sin(\psi) & \cos(\psi)\cos(\phi) & -\cos(\psi)\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi)\end{bmatrix}\end{aligned} \tag{4.8}$$

4.2.4. From BODY to CAM

Transforming from BODY to CAM frame, we first need to rotate from BODY back to coordinate frame 2, i.e. a rotation of $90\,^\circ$ about the z_b axis. When

$$\begin{aligned}\boldsymbol{A}_3 &= \boldsymbol{R}_{z,\alpha-\frac{\pi}{2}}\boldsymbol{R}_{x,-\frac{\pi}{2}} \\ \boldsymbol{A}_4 &= \boldsymbol{R}_{z,\beta}\boldsymbol{R}_{x,\frac{\pi}{2}},\end{aligned}$$

the transformation from BODY to CAM becomes

$$\begin{aligned} \boldsymbol{R}_c^b &= \boldsymbol{R}_{z,\frac{\pi}{2}} \boldsymbol{A}_3 \boldsymbol{A}_4 \\ \boldsymbol{R}_c^b &= \begin{bmatrix} \cos(\alpha)\cos(\beta) & -\sin(\alpha) & \cos(\alpha)\sin(\beta) \\ \sin(\alpha)\cos(\beta) & \cos(\alpha) & \sin(\alpha)\cos(\beta) \\ \sin(\beta) & 0 & -\cos(\beta) \end{bmatrix} \end{aligned} \tag{4.9}$$

4.3. Flight Dynamics

A model for the flight dynamics of the UAV is needed in the MPC. In order to describe the UAV's kinematics completely, 6 degrees of freedom (DOFs) are needed, three for position and three for orientation. This can be found in Fossen [2011] and looks generally like

$$\dot{\boldsymbol{p}}_{b/n}^n = \boldsymbol{R}_b^n(\boldsymbol{\Theta}_{nb})\boldsymbol{v}_{b/n}^b \tag{4.10}$$

$$\dot{\boldsymbol{\Theta}}_{nb} = \boldsymbol{T}_\Theta(\boldsymbol{\Theta}_{nb})\boldsymbol{\omega}_{b/n}^b \tag{4.11}$$

where $\boldsymbol{p}_{b/n}^n = [N, E, D]^\top$,$\boldsymbol{\Theta}_{nb} = [\phi, \theta, \psi]^\top$, $\boldsymbol{v}_{b/n}^b = [u, v, w]^\top$ is the linear velocity of the UAV relative to the NED frame decomposed in the BODY frame, and $\boldsymbol{\omega}_{b/n}^b = [p, q, r]^\top$ is the angular velocity of the UAV relative to the NED frame decomposed in the BODY frame. $\boldsymbol{T}_\Theta(\boldsymbol{\Theta}_{nb})$ is a rotation matrix relating the angular velocity in the BODY frame to the NED frame. For reasons that will be shown shortly, it will not be used here.

4.3.1. Linear Velocity

Assuming that the Piccolo autopilot maintains a constant true airspeed in the x_b-direction, the ground speed, V_g, can be estimated by adding the wind estimates and the true airspeed. This can be seen in Figure 4.2, and we have that

$$\vec{V}_g = \vec{V}_t + \vec{V}_w \tag{4.12}$$

where $\vec{V}_t$ is the true airspeed and $\vec{V}_w$ is the wind estimate. The wind estimate is supplied by the Piccolo autopilot and is decomposed as $\boldsymbol{V}_w = [V_{w,N}, V_{w,E}, V_{w,D}]$.

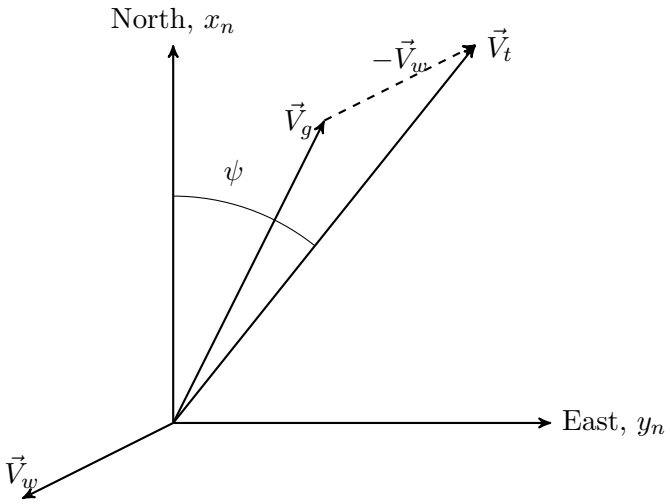


Figure 4.2.: Description of the relationship between ground velocity, wind velocity and true air velocity.

In order to describe the linear velocities, we insert (4.8) into (4.10) and add the wind estimates. Since the Piccolo autopilot is assumed to handle vertical motion, $\dot{D} = 0$.

$$\dot{N} = V\cos(\psi) + V_{w,N} \tag{4.13a}$$

$$\dot{E} = V\sin(\psi) + V_{w,E} \tag{4.13b}$$

$$\dot{D} = 0 \tag{4.13c}$$

4.3.2. Angular Velocity

Now, for the angular velocities, we have already assumed that $\theta = \dot{\theta} = 0$ and that $\dot{\phi}$ is our control input. Using this, only $\dot{\psi}$ needs to be found. By using perturbation theory, i.e. linearising the dynamics, and assuming that the aircraft is trimmed so that the angle of attack is zero, Fossen [2013] finds that the yaw rate

$$\dot{\psi}^n = r = \frac{g}{V}\sin(\phi) \overset{\text{small } \phi}{\approx} \frac{g}{V}\phi \tag{4.14}$$

The approximation $\sin(\phi) \overset{\text{small } \phi}{\approx} \phi$ is a commonly used approximation called the *small-angle approximation*. As all angular velocities now have been found, the rotation matrix $\boldsymbol{T}_\Theta(\boldsymbol{\Theta}_{nb})$ in (4.11) does not have to be found.

To summarize, the kinematics describing the system are

$$\dot{N} = V\cos(\psi) + V_{w,N} \tag{4.15a}$$

$$\dot{E} = V\sin(\psi) + V_{w_E} \tag{4.15b}$$

$$\dot{D} = 0 \tag{4.15c}$$

$$\dot{\phi} = u_\phi \tag{4.15d}$$

$$\dot{\theta} = 0 \tag{4.15e}$$

$$\dot{\psi} = \frac{g}{V}\phi \tag{4.15f}$$

5. Control Formulation

In this chapter, the control formulations used in this thesis are stated explicitly. The numerical values of MPC parameters are stated and explained. For simplicity, the MPC equations (2.11) are restated:

$$\min_{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)} \int_{t_0}^{t_0+T} \|\boldsymbol{h}(t,\boldsymbol{x}(t),\boldsymbol{u}(t)) - \boldsymbol{\eta}(t)\|^2_{\boldsymbol{Q}} dt + \|\boldsymbol{m}(\boldsymbol{x}(t_0+T), t_0+T) - \boldsymbol{\mu}\|^2_{\boldsymbol{R}} \tag{5.1a}$$

$$s.t.$$

$$0 = \boldsymbol{f}(t,\boldsymbol{x}(t),\dot{\boldsymbol{x}}(t),\boldsymbol{u}(t)) \tag{5.1b}$$

$$0 \geq \boldsymbol{s}(t,\boldsymbol{x}(t),\boldsymbol{u}(t)) \tag{5.1c}$$

$$0 = \boldsymbol{r}(\boldsymbol{x}(t_0+T),(t_0+T)) \tag{5.1d}$$

The prediction horizon T is the same in all control methods. As mentioned in Section 2.3, the prediction horizon must be long enough to include the interesting behaviour or so long that the system converges within it. Now, setting the prediction horizon sufficiently long to allow the UAV to reach the object within it, will not always be possible as one might start far away. Instead, the prediction horizon is set to the length of circling the object for one whole round. Setting a prediction horizon that is much shorter than the duration of one round of circling may cause the controller to steer the UAV directly towards the object, not realizing that it should start circling before it is too late. Setting the prediction horizon much longer than the duration of one round of circling will cause the controller to be slower, and it may also lead to a less optimal flight path. If the prediction horizon for example is set to 1.5 rounds of circling, the controller will attempt to minimize the distance between the UAV and the object twice as hard for the first half of the circle as for the second half. Below, this prediction horizon is found:

$$O = 2\pi r$$

where O is the circumference of the circle and r is the radius of the circle. The radius of the circle can be found using the relationship between angular speed, ω, and linear speed, v

$$v = r\omega$$
$$r = \frac{v}{\omega}$$

Now, an expression for the prediction horizon can be found by

$$O = vT$$
$$T = \frac{O}{v} = \frac{2\pi r}{v} = \frac{2\pi \frac{v}{\omega}}{v} = \frac{2\pi}{\omega}$$

Assuming that the maximum roll rate of $\phi_{max} = \frac{30\pi}{180} \approx 0.52$ is used, which is explained below, the prediction horizon can be found

$$T = \frac{2\pi}{\dot{\psi}_{max}} = \frac{2\pi}{\frac{g}{v}\phi_{max}} = \frac{2\pi v}{g\phi_{max}} \approx 35s \tag{5.2}$$

where $v = 28\frac{m}{s}$ and $g = 9.81\frac{m}{s^2}$.

5.1. Control Method 1

Control Method 1 (CM1) is the simplest and most intuitive of the control methods. CM1 controls the UAV by minimizing the distance from the UAV to the object, and the desired gimbal angles are found explicitly and handed to the gimbal.

In order to find the gimbal angles that would make the camera point directly at the object, the vector from the UAV to the object decomposed in the BODY frame needs to be found:

$$\mathbf{p}^b_{o/b} = \mathbf{R}^b_n \mathbf{p}^n_{o/b} = \begin{bmatrix} d^b_x \\ d^b_y \\ d^b_z \end{bmatrix} \tag{5.3}$$

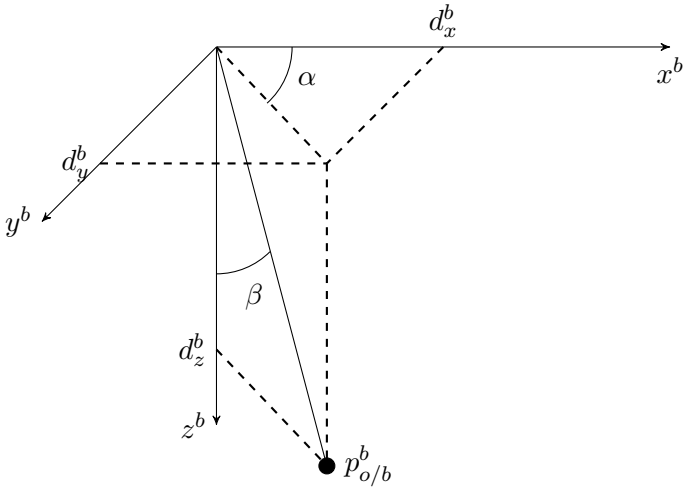


Figure 5.1.: Pan and tilt angles from the BODY frame

This is shown in figure 5.1, and the gimbal angles can now be found by

$$\alpha_d = \left\{ \begin{array}{ll} \alpha_{\min} & , atan2(d_y^b, d_x^b) < \alpha_{\min} \\ \alpha_{\max} & , atan2(d_y^b, d_x^b) > \alpha_{\max} \\ atan2(d_y^b, d_x^b) & , \text{otherwise} \end{array} \right\} \tag{5.4a}$$

$$\beta_d = \left\{ \begin{array}{ll} \beta_{\min} & , atan2\left(\sqrt{{d_x^b}^2 + {d_y^b}^2}, d_z^b\right) < \beta_{\min} \\ \beta_{\max} & , atan2\left(\sqrt{{d_x^b}^2 + {d_y^b}^2}, d_z^b\right) > \beta_{\max} \\ atan2\left(\sqrt{{d_x^b}^2 + {d_y^b}^2}, d_z^b\right) & , \text{otherwise} \end{array} \right\} \tag{5.4b}$$

where α_d is the desired pan angle, β_d is the desired tilt angle, and $atan2(y, x)$ is an arctangent operator that returns the angle in the appropriate quadrant of the point (x, y).

The MPC formulation for CM1 is:

$$\boldsymbol{h}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) = \begin{bmatrix} N \\ E \\ u_\phi \end{bmatrix}, \boldsymbol{\eta} = \begin{bmatrix} N_{obj} \\ E_{obj} \\ 0 \end{bmatrix} \tag{5.5a}$$

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10^7 \end{bmatrix} \tag{5.5b}$$

$$\boldsymbol{m} = 0 \tag{5.5c}$$

$$\dot{N} = V\cos(\psi) + V_{w,N} \tag{5.5d}$$

$$\dot{E} = V\sin(\psi) + V_{w,E} \tag{5.5e}$$

$$\dot{V}_{w,N} = 0 \tag{5.5f}$$

$$\dot{V}_{w,E} = 0 \tag{5.5g}$$

$$\dot{\phi} = u_\phi \tag{5.5h}$$

$$\dot{\psi} = \frac{g}{V}\phi \tag{5.5i}$$

$$-\frac{30\pi}{180} \leq \phi \leq \frac{30\pi}{180} \tag{5.5j}$$

$$-0.1 \leq u_\phi \leq 0.1 \tag{5.5k}$$

The weight matrix values is $\boldsymbol{Q} = diag([q_1, q_2, q_3])$ with weight parameters $q_1 = 1$, $q_2 = 1$, and $q_3 = 10^7$. q_1 and q_2 are equal since displacement in north and east direction should be penalized equally. They are both set to 1 since the magnitude of the weighting parameters is only important relative to each other, and 1 is a nice starting point. The value of q_3 was determined by trial and error, and $q_3 = 10^7$ made the UAV find a smooth roll angle trajectory when approaching the object.

The roll angle trajectory was, of course, also influenced by the limitation in equation (5.5k). The reason for both punishing and limiting the roll rate is bipartite. As mentioned, the value of q_3 was found such that the behaviour when closing in on an object is smooth. Thus, such a value of q_3 will make $q_3 u_\phi^2$ insignificant compared to $q_1(N - N_{obj})^2 + q_2(E - E_{obj})^2$ when the UAV is far away from the object as the terms are quadratic. In order for the system to not misbehave when far away from the object, a hard constraint on the roll angle is set. The value of $-u_{\phi,min} = u_{\phi,max} = 0.1$ was found by trial and error.

The constraint on the roll angle in equation (5.5j) was determined by the pilots assisting in the execution of this project.

5.2. Control Method 2

Control Method 2 (CM2) is based on the principle that the camera is to point parallel to the vector from the UAV to the object on the ground. Both the vector from UAV to object, (5.6), and the camera vector, (5.8), are found as unit vectors, and the difference between them is object for minimization.

$$\boldsymbol{p}^n_{o/b} = \begin{bmatrix} N_{obj} - N \\ E_{obj} - E \\ D_{obj} - D \end{bmatrix} = \begin{bmatrix} x^n_{o/b} \\ y^n_{o/b} \\ z^n_{o/b} \end{bmatrix}$$

$$\bar{\boldsymbol{p}}^n_{o/b} = \frac{1}{\|\boldsymbol{p}^n_{o/b}\|}\boldsymbol{p}^n_{o/b} = \frac{1}{\sqrt{{x^n_{o/b}}^2 + {y^n_{o/b}}^2 + {z^n_{o/b}}^2}} \begin{bmatrix} x^n_{o/b} \\ y^n_{o/b} \\ z^n_{o/b} \end{bmatrix} = \begin{bmatrix} \bar{x}^n_{o/b} \\ \bar{y}^n_{o/b} \\ \bar{z}^n_{o/b} \end{bmatrix} \tag{5.6}$$

$$\bar{\boldsymbol{p}}^n_{cam} = \boldsymbol{R}^n_c \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x^n_{cam} \\ y^n_{cam} \\ z^n_{cam} \end{bmatrix} \tag{5.7}$$

$$\bar{\boldsymbol{p}}^n_{cam} = \begin{bmatrix} (-\sin(\psi)\cos(\phi)\sin(\alpha) + \cos(\psi)\cos(\alpha))\sin(\beta) + \sin(\psi)\sin(\phi)\cos(\beta) \\ (\cos(\psi)\cos(\phi)\sin(\alpha) + \sin(\psi)\cos(\alpha))\sin(\beta) - \cos(\psi)\sin(\phi)\cos(\beta) \\ \sin(\phi)\sin(\alpha)\sin(\beta) + \cos(\phi)\cos(\beta) \end{bmatrix} \tag{5.8}$$

$$\boldsymbol{h}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) = \begin{bmatrix} N \\ E \\ \bar{x}^n_{o/b} - x^n_{cam} \\ \bar{y}^n_{o/b} - y^n_{cam} \\ \bar{z}^n_{o/b} - z^n_{cam} \\ u_\phi \\ s_\alpha \\ s_\beta \end{bmatrix}, \boldsymbol{\eta} = \begin{bmatrix} N_{obj} \\ E_{obj} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.9a}$$

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10^8 \end{bmatrix} \tag{5.9b}$$

$$\dot{N} = V\cos(\psi) + V_{w,N} \tag{5.9c}$$
$$\dot{E} = V\sin(\psi) + V_{w_E} \tag{5.9d}$$
$$\dot{D} = 0 \tag{5.9e}$$
$$\dot{N}_{obj} = 0 \tag{5.9f}$$
$$\dot{E}_{obj} = 0 \tag{5.9g}$$
$$\dot{V}_{w,N} = 0 \tag{5.9h}$$
$$\dot{V}_{w,E} = 0 \tag{5.9i}$$
$$\dot{\phi} = u_\phi \tag{5.9j}$$
$$\dot{\psi} = \frac{g}{V}\phi \tag{5.9k}$$
$$\dot{\alpha} = u_\alpha \tag{5.9l}$$
$$\dot{\beta} = u_\beta \tag{5.9m}$$

$$-\frac{30\pi}{180} \leq \phi \leq \frac{30\pi}{180} \tag{5.9n}$$
$$-0.1 \leq u_\phi \leq 0.1 \tag{5.9o}$$
$$-\pi \leq \alpha + s_\alpha \leq \pi \tag{5.9p}$$
$$\frac{10\pi}{180} \leq \beta + s_\beta \leq \frac{\pi}{2} \tag{5.9q}$$
$$-\pi \leq u_\alpha \leq \pi \tag{5.9r}$$
$$-\frac{\pi}{2} \leq u_\beta \leq \frac{\pi}{2} \tag{5.9s}$$

(5.9h) and (5.9i) are included in order to pass the wind estimates $V_{w,N}$ and $V_{w,E}$ to the controller. Slack variables are added to the pan and tilt constraints (5.9p) and (5.9q)

because it helped with some infeasibility issues encountered with CM3. Since CM2 has a similar functionality and level of complexity, they are added here as well. The CM2 simulation was never run as discussed in Chapter 7 and 9, so it is only assumed that they are needed here.

5.3. Control Method 3

Control Method 3 (CM3) is based on the principle that the point on the ground that the center of the camera lens points at, denoted camera ground point (CGP), should be as close as possible to the the object on the ground.

$$\boldsymbol{p}^n_{cgp/b} = \boldsymbol{R}^n_c \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix} \tag{5.10}$$

where d is the length of the vector from the UAV to the CGP. The z-element in $\boldsymbol{p}^n_{cgp/b}$ is assumed known because the UAV's height over ground is known. When $z^n_{cgp/b}$ is known in

$$\boldsymbol{p}^n_{cgp/b} = \begin{bmatrix} x^n_{cgp/b} \\ y^n_{cgp/b} \\ z^n_{cgp/b} \end{bmatrix} = \begin{bmatrix} ((-\sin(\psi)\cos(\phi)\sin(\alpha) + \cos(\psi)\cos(\alpha))\sin(\beta) + \sin(\psi)\sin(\phi)\cos(\beta))d \\ ((\cos(\psi)\cos(\phi)\sin(\alpha) + \sin(\psi)\cos(\alpha))\sin(\beta) - \cos(\psi)\sin(\phi)\cos(\beta))d \\ (\sin(\phi)\sin(\alpha)\sin(\beta) + \cos(\phi)\cos(\beta))d \end{bmatrix} \tag{5.11}$$

d can be found as

$$d = \frac{z^n_{cgp/b}}{\sin(\phi)\sin(\alpha)\sin(\beta) + \cos(\phi)\cos(\beta)} \tag{5.12}$$

From this, $x^n_{cgp/b}$ and $y^n_{cgp/b}$ can be found by inserting (5.12) into (5.11):

$$x^n_{cgp/b} = ((-\sin(\psi)\cos(\phi)\sin(\alpha) + \cos(\psi)\cos(\alpha))\sin(\beta) + \sin(\psi)\sin(\phi)\cos(\beta))d \tag{5.13}$$

$$y^n_{cgp/b} = ((\cos(\psi)\cos(\phi)\sin(\alpha) + \sin(\psi)\cos(\alpha))\sin(\beta) - \cos(\psi)\sin(\phi)\cos(\beta))d \tag{5.14}$$

The position of the CGP relative to the NED reference frame $\{n\}$ is

$$\begin{bmatrix} x^n_{cgp/n} \\ y^n_{cgp/n} \\ z^n_{cgp/n} \end{bmatrix} = \begin{bmatrix} x^n_{cgp/b} \\ y^n_{cgp/b} \\ z^n_{cgp/b} \end{bmatrix} + \begin{bmatrix} x^n_{b/n} \\ y^n_{b/n} \\ z^n_{b/n} \end{bmatrix} = \begin{bmatrix} x^n_{cgp/b} \\ y^n_{cgp/b} \\ z^n_{cgp/b} \end{bmatrix} + \begin{bmatrix} N \\ E \\ D \end{bmatrix} \tag{5.15}$$

Since $z^n_{cgp/b} = -D$, it follows that $z^n_{cgp/n} = 0$. Minimizing the distance between the camera ground point and the object position can then be written as

$$\begin{bmatrix} x^n_{cgp/b} \\ y^n_{cgp/b} \end{bmatrix} + \begin{bmatrix} N \\ E \end{bmatrix} \rightarrow \begin{bmatrix} N_{obj} \\ E_{obj} \end{bmatrix} \tag{5.16}$$

$$\implies \begin{bmatrix} x^n_{cgp/b} \\ y^n_{cgp/b} \end{bmatrix} \rightarrow \begin{bmatrix} N_{obj} \\ E_{obj} \end{bmatrix} - \begin{bmatrix} N \\ E \end{bmatrix} \tag{5.17}$$

$$\boldsymbol{h}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) = \begin{bmatrix} N \\ E \\ N + x^n_{cgp/b} \\ E + y^n_{cgp/b} \\ u_\phi \\ s_\alpha \\ s_\beta \end{bmatrix}, \boldsymbol{\eta} = \begin{bmatrix} N_{obj} \\ E_{obj} \\ N_{obj} \\ E_{obj} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.18a}$$

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 800 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 800 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^8 \end{bmatrix} \tag{5.18b}$$

$$\dot{N} = V\cos(\psi) + V_{w,N} \tag{5.18c}$$
$$\dot{E} = V\sin(\psi) + V_{w,E} \tag{5.18d}$$
$$\dot{D} = 0 \tag{5.18e}$$
$$\dot{V}_{w,N} = 0 \tag{5.18f}$$
$$\dot{V}_{w,E} = 0 \tag{5.18g}$$
$$\dot{\phi} = u_\phi \tag{5.18h}$$
$$\dot{\psi} = \frac{g}{V}\phi \tag{5.18i}$$
$$\dot{\alpha} = u_\alpha \tag{5.18j}$$
$$\dot{\beta} = u_\beta \tag{5.18k}$$

$$-\frac{30\pi}{180} \leq \phi \leq \frac{30\pi}{180} \tag{5.18l}$$
$$-0.1 \leq u_\phi \leq 0.1 \tag{5.18m}$$
$$-\pi \leq \alpha + s_\alpha \leq \pi \tag{5.18n}$$
$$\frac{10\pi}{180} \leq \beta + s_\beta \leq \frac{\pi}{2} \tag{5.18o}$$
$$-\pi \leq u_\alpha \leq \pi \tag{5.18p}$$
$$-\frac{\pi}{2} \leq u_\beta \leq \frac{\pi}{2} \tag{5.18q}$$

The weight parameter values of $q_1 = q_2 = 1$ and $q_5 = 10^7$, i.e. weight on roll rate, has been kept since only the relative magnitude is important. Therefore, only q_3, q_4, q_6, q_7 need to be set.

q_3 will be equal to q_4 because punishing eastwards or northwards deviation differently makes no sense. q_3 and q_4 were found by trial and error, and a value of $q_3 = q_4 = 800$ was shown to give good results in simulations.

q_6 and q_7 were also found by trial and error, and a value of $q_6 = q_7 = 10^8$ were found to give a good results in simulations.

Slack variables are added to the pan and tilt constraints, 5.18n and 5.18o, because it was experienced to help with some infeasibility issues encountered in ACADO.

6. Software implementation

6.1. ACADO

The ACADO implementation of control methods 1, 2, and 3 are shown in detail in Appendix A.2, A.3, and A.4, respectively. When running the executable that these programs creates, one gets a folder called "export" with a static library and some header files. These are included in the DUNE project in order to access the functions solving the MPC problem.

6.2. DUNE

In order to implement the trajectory planner on the UAV, it must be implemented as a DUNE task. It is built as a simple state machine, resting in an idle state until it receives a message telling it to start. Then it must reply that it has received the message and started operating. This will be handled by an initialization state. From the initialization state it will automatically transition into a state in which trajectories are planned and desired states are output. From this state, it can receive a message telling it to quit its operation and return to the idle state again. This is summarized in pseudocode in Algorithm 1.

During operation, the following messages may be received by the MPC module:

- *Estimated State* - contains latitude and longitude coordinates of a DUNE specified NED reference frame and the UAV's x, y, and z offsets relative to that frame. It also contains roll, pitch, and yaw estimates of the UAV. This message is dispatched from the Piccolo approximately six times per second
- *Estimated Stream Velocity* - contains wind estimate decomposed in the NED reference frame that is given in the estimated state message

- *Run message* - contains a text string which is "Activate" when control is handed to the MPC and "Deactivate" when control is taken from the MPC

During operation, the DUNE task may send the following messages:

- *Reply Run Message* - reply to a Run message which confirms that the MPC is activated or deactivated
- *Desired Roll* - sends the desired roll angle found by the MPC to the Piccolo.
- *Control by Roll Message* - In order for the Piccolo to accept the desired roll message and track the roll angle received, this message has to be sent with a flag set to high. When exiting and giving back control, this message has to be sent with the same flag set to low, so that the Piccolo no longer accepts roll control messages.

Algorithm 1 DUNE task

```
while !stopping() do
  switch (state)
  case IDLE:
      Wait for messages
      Handle messages if received
      break;
  case INIT:
      Confirm that activation message was received
      Tell Piccolo to accept roll control messages
      state = RUN
      break;
  case RUN:
      Run MPC algorithm
      Send desired roll, pan, and tilt angles
      Handle messages if received
      break;
  case EXIT:
      Confirm that deactivate message was received
      Tell Piccolo to no longer accept roll control messages
      state = IDLE
      break;
  end switch
end while
```

7. Simulation

7.1. Simulation Setup

The simulation framework is simple and written in C. The flight model used is the same one as in the controller, which makes the simulations useful for tuning of the MPC weight and constraint parameters. These were stated in Chapter 5, and were found by trial and error.

The discretized version of the flight model used in the controller is applied in the simulation model:

$$N_{i+1} = N_i + (V\cos(\psi)_i + V_{w,N})\,\mathrm{d}t \tag{7.1a}$$
$$E_{i+1} = E_i + (V\sin(\psi) + V_{w,E})\,\mathrm{d}t \tag{7.1b}$$
$$D_{i+1} = D_i \tag{7.1c}$$
$$\phi_{i+1} = \phi_i + u_\phi\,\mathrm{d}t \tag{7.1d}$$
$$\theta_{i+1} = \theta_i = 0 \tag{7.1e}$$
$$\psi_{i+1} = \psi_i + \frac{g}{V}\phi_i\,\mathrm{d}t \tag{7.1f}$$
$$\alpha_{i+1} = \alpha_i + u_\alpha\,\mathrm{d}t \tag{7.1g}$$
$$\beta_{i+1} = \beta_i + u_\beta\,\mathrm{d}t \tag{7.1h}$$

The length of the simulations is 200 seconds, and the sampling time is $\mathrm{d}t = 0.001$. For CM2 and CM3, the pan and tilt control inputs, u_α and u_β, are found explicitly by the MPC. For CM1 the control inputs $u_\alpha = \pi(-\alpha_i + \alpha_{d,i})$ and $u_\beta = \frac{\pi}{2}(-\beta_i + \beta_{d,i})$ are used, where $\alpha_{d,i}$ and $\beta_{d,i}$ are found by equation 5.4a and 5.4b, respectively. The MPC algorithm is run once every second, and the control input from the MPC is applied to the model for one second. The two operations, i.e. running the controller every

second and updating the model 1000 times per second, are not performed in parallel in two separate threads, which would be the most realistic scenario. Instead, the model "pauses" when the controller is running, and "unpauses" when a control input is ready. This is chosen for simplicity, and because the more realistic alternative will be tested in the HIL simulations. The only real difference between the two alternatives is that when run in parallel, it can start calculating a new control input as soon as it is done with the previous one. As long as one second is not unrealistically often, this alternative will, if anything, be a pessimistic approximation.

For all simulations, unless otherwise is noted, the initial values are

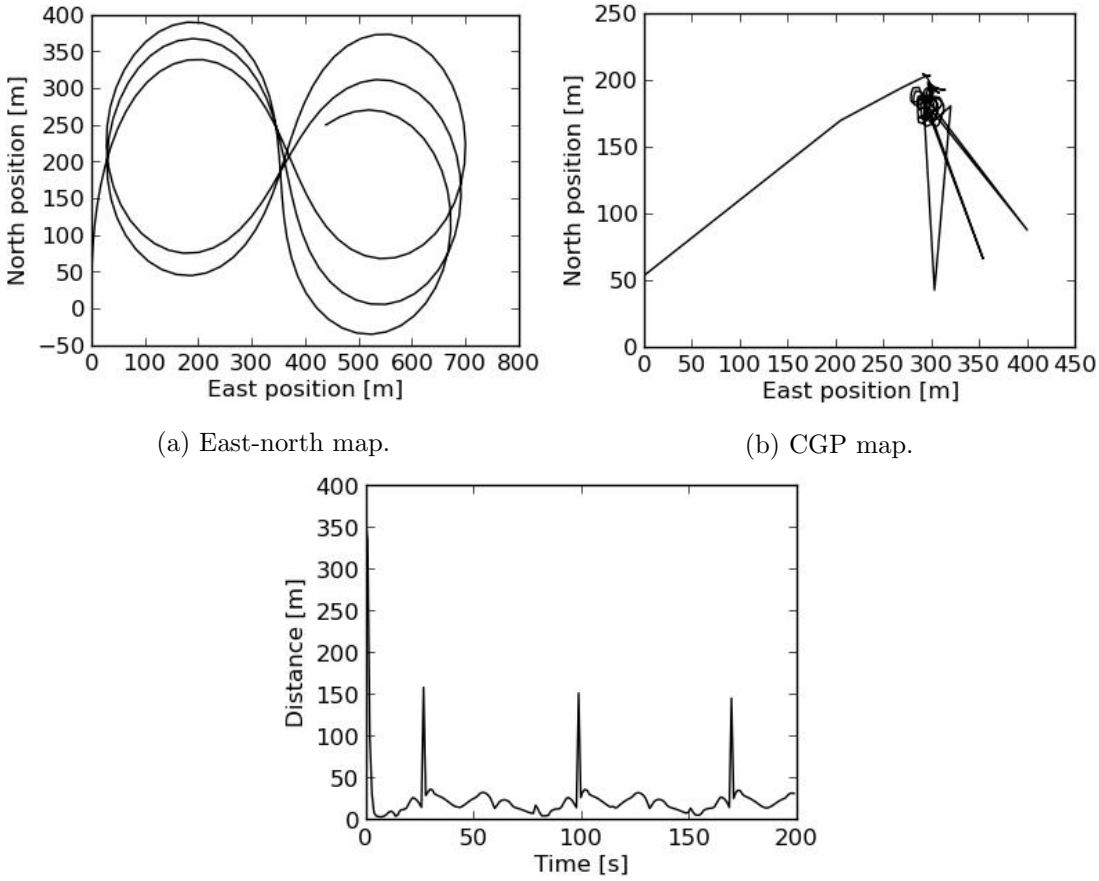
- Object's position - $[N_{obj,0}, E_{obj,0}, D_{obj,0}] = [200, 300, 0]$
- UAV's position - $[N_0, E_0, D_0] = [0, 0, -300]$
- UAV's attitude - $[\phi_0, \theta_0, \psi_0] = [0, 0, 0]$
- Gimbal's attitude - $[\alpha_0, \beta_0] = [0, \frac{10\pi}{180}]$.

7.2. Simulation Results

In this section, the results of the simulations will be displayed. The scenarios that are simulated are without and with wind.

7.2.1. Without Disturbances

In this section, all control methods are simulated with wind $V_{w,N} = V_{w,E} = 0$.



(a) East-north map.

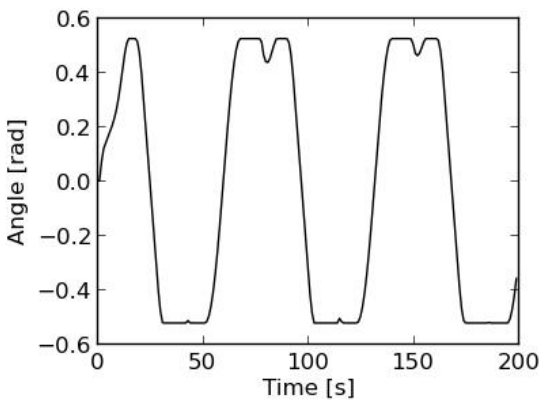
(b) CGP map.

(c) Difference between CGP and object position.

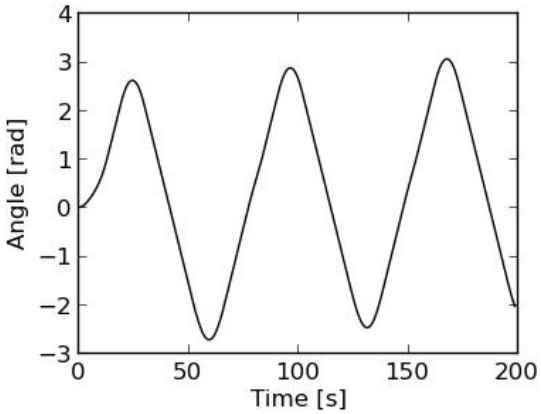
Figure 7.1.: CM1 behaviour without disturbances.

CM1

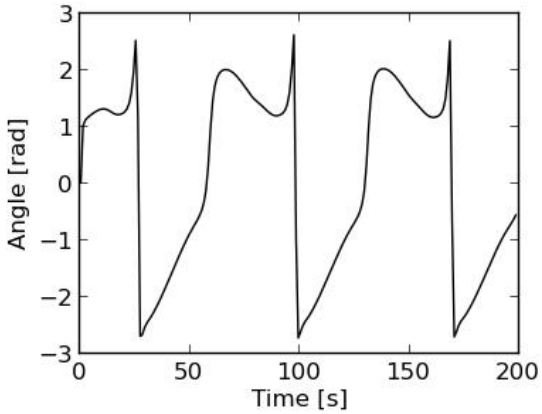
The simulation results without wind for CM1 can be seen in figure 7.1 and 7.2. This is not expected behaviour of the UAV as the easiest solution should be to loiter in a circular pattern above the object, and something seems to be faulty. The problem seems to be that the MPC algorithm attempts to minimize the heading value. A large heading offset of -50π is added in order to fix this problem, and the results can be viewed in figure 7.3 and 7.4. This is discussed further in Section 9.1.1.



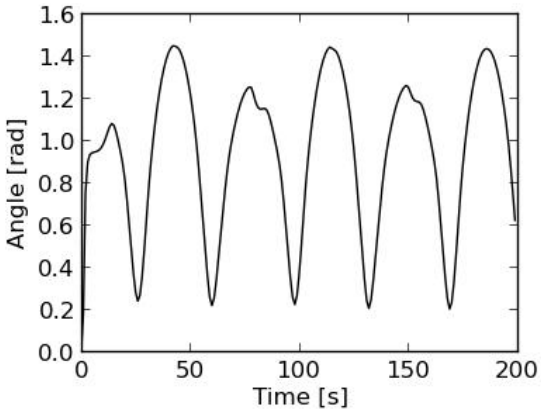
(a) Roll angle.



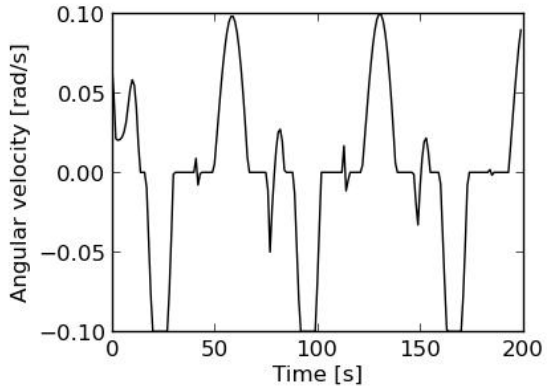
(b) Heading angle.



(c) Pan angle.

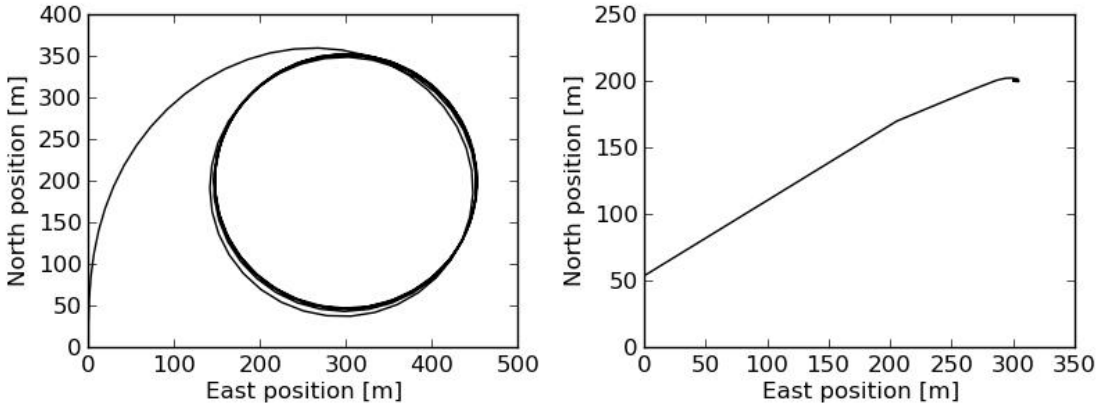


(d) Tilt angle.



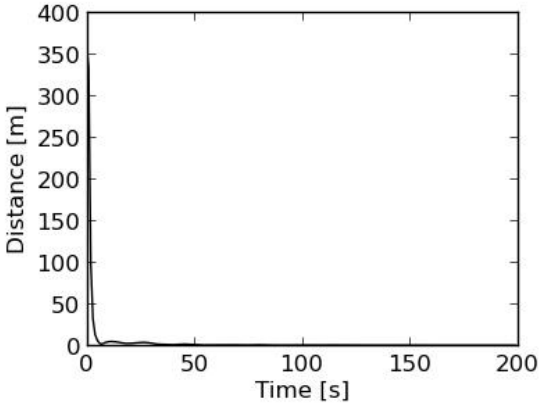
(e) Roll control input.

Figure 7.2.: CM1 angles and control variable without disturbances.



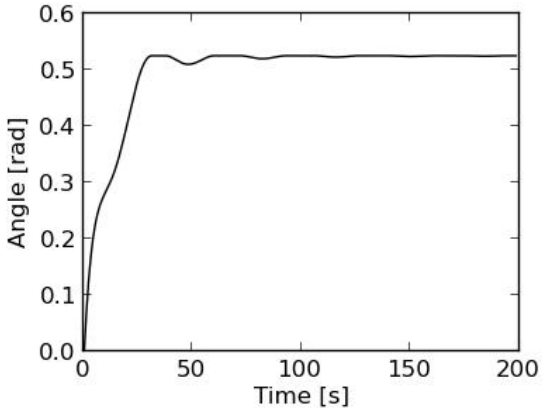
(a) East-north map.

(b) CGP map.

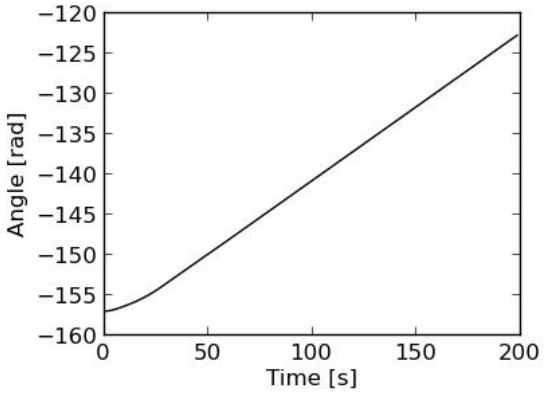


(c) Difference between CGP and object position.

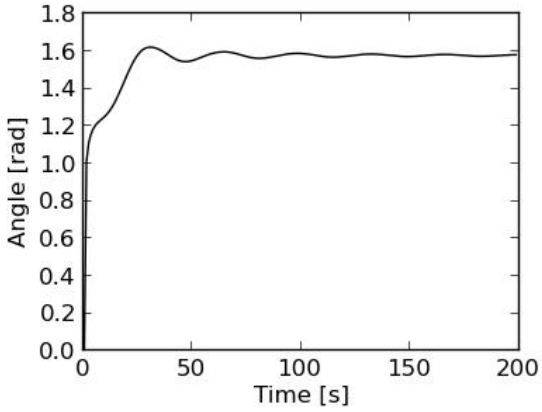
Figure 7.3.: CM1 behaviour without disturbances and with -50π offset on heading.



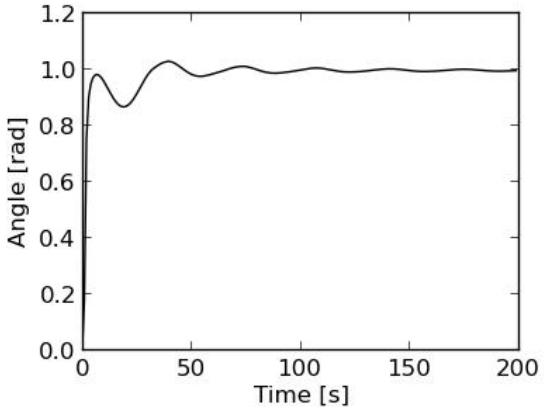
(a) Roll angle.



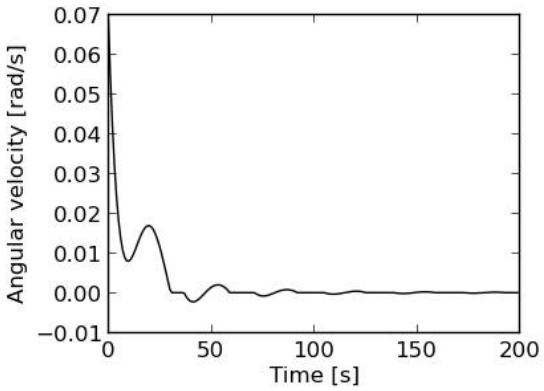
(b) Heading angle.



(c) Pan angle.



(d) Tilt angle.

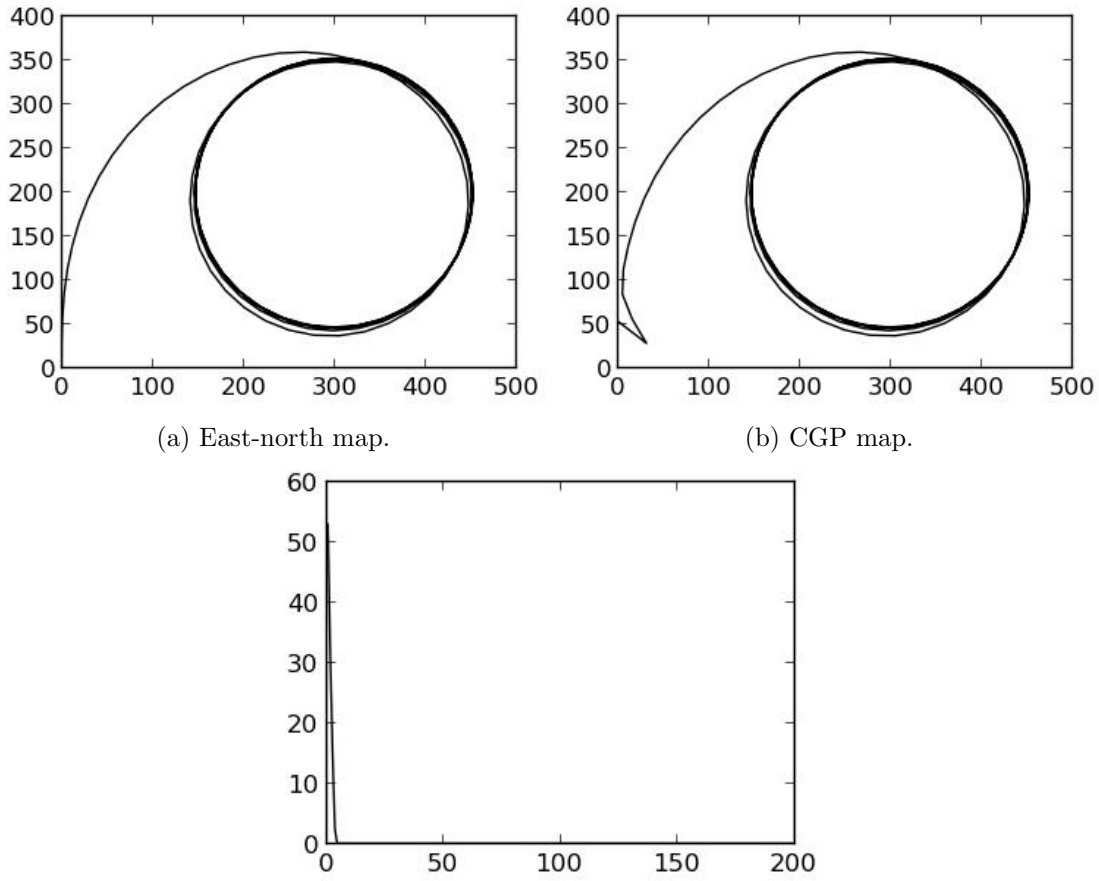


(e) Roll control input.

Figure 7.4.: CM1 angles and control variable without disturbances and with -50π offset on heading.

CM2

Running the simulations control method 2 failed as all variables were set to NaN by ACADO immediately after running the control algorithm. In order to show what may cause this, another simulation scenario is run. The controller is based on the same equations, but the PTG is controlled to point straight down, i.e. parallel to the z_n vector. The results of this can be seen in figure 7.5.



(a) East-north map.

(b) CGP map.

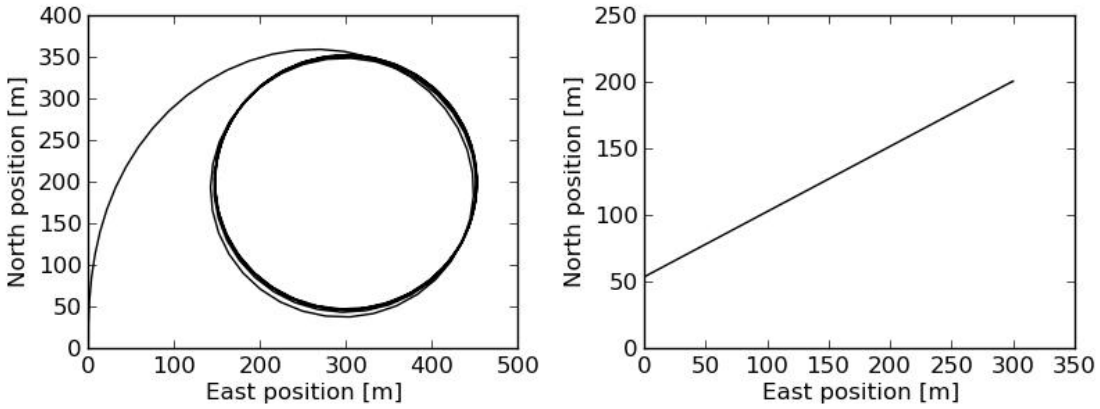
(c) Difference between CGP and UAV position.

Figure 7.5.: Modified CM2 behaviour without disturbances.

Because of this problem, the CM2 controller will be excluded from further testing.

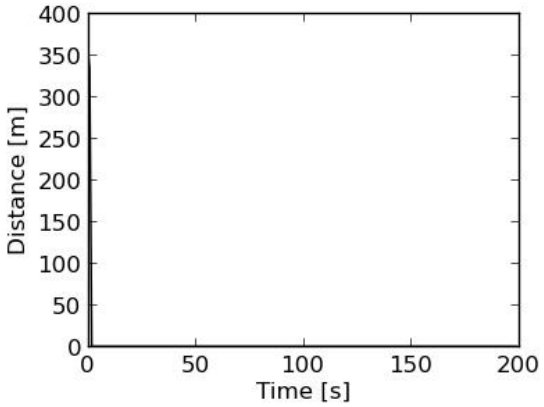
CM3

Figure 7.6 and 7.7 shows the simulation without disturbances with the CM3 controller.



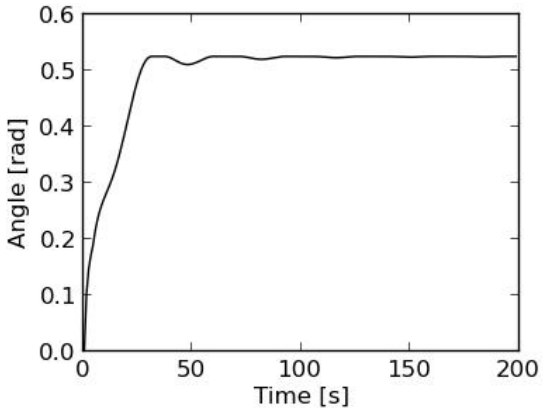
(a) East-north map.

(b) CGP map.

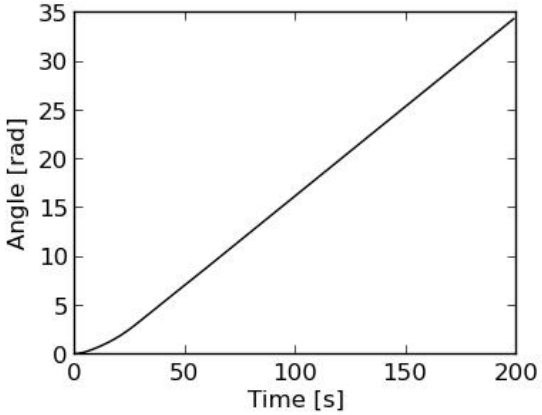


(c) Difference between CGP and object position.

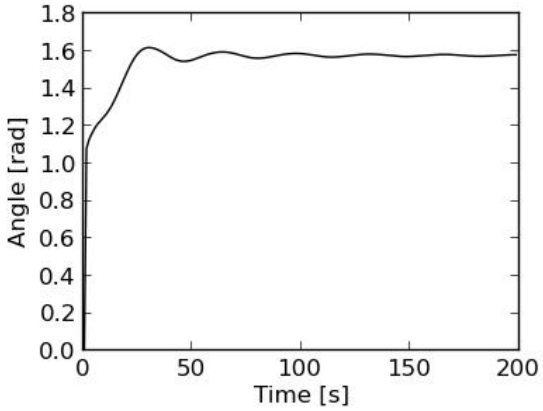
Figure 7.6.: CM3 behaviour without disturbances.



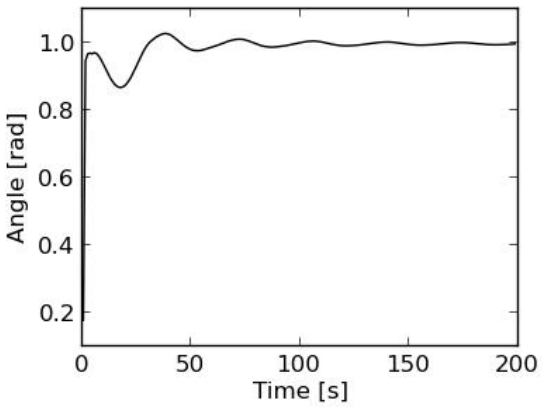
(a) Roll angle.



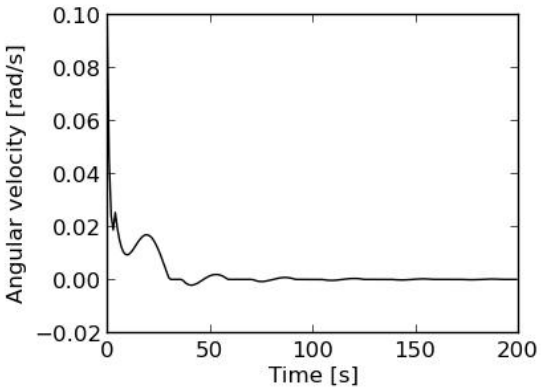
(b) Heading angle.



(c) Pan angle.



(d) Tilt angle.



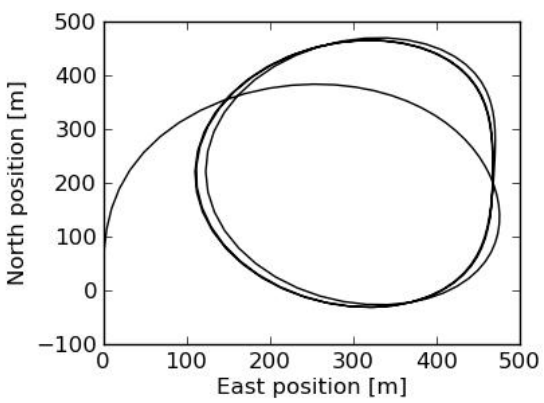
(e) Roll control input.

Figure 7.7.: CM3 angles and control variable without disturbances.

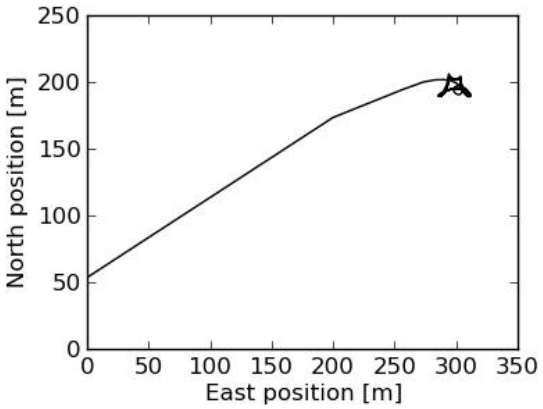
7.2.2. With Wind

CM1

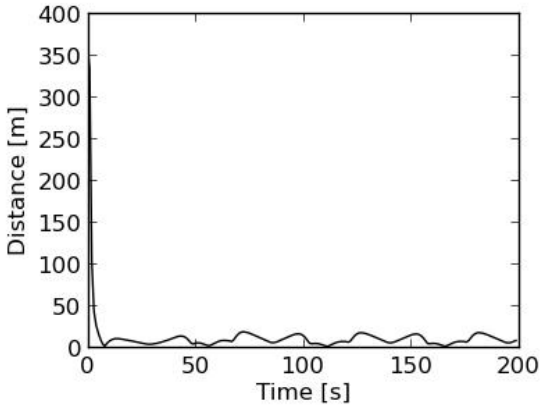
Figure 7.8 and 7.9 shows the simulation with wind with CM1.



(a) East-north map.

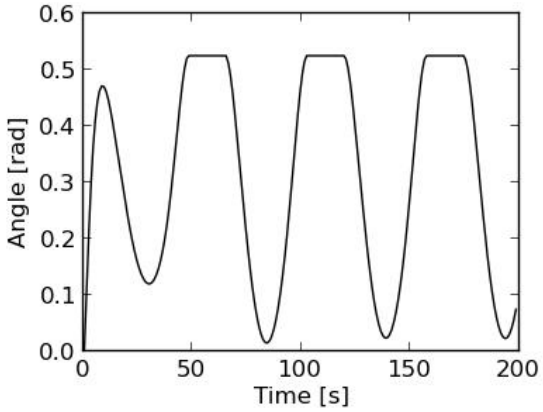


(b) CGP map.

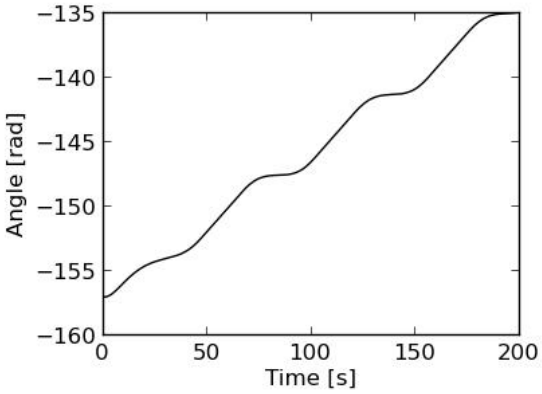


(c) Difference between CGP and object position.

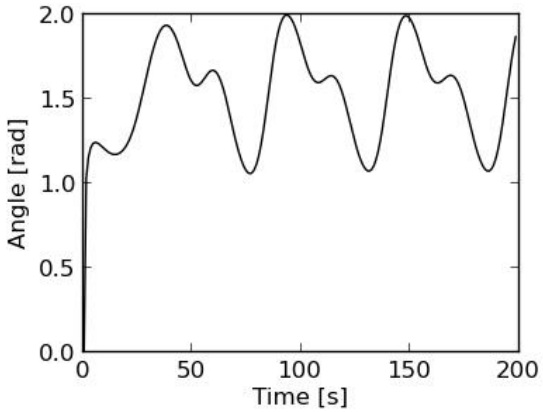
Figure 7.8.: CM1 behaviour with wind disturbance.



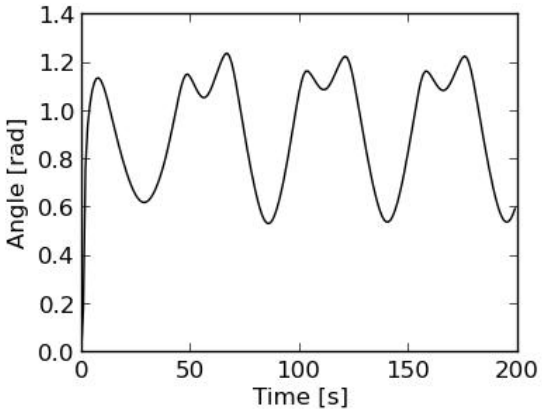
(a) Roll angle.



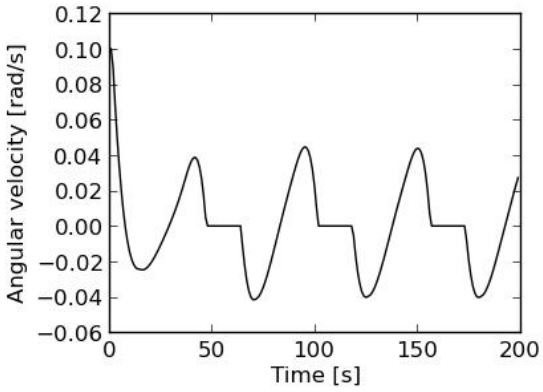
(b) Heading angle.



(c) Pan angle.



(d) Tilt angle.

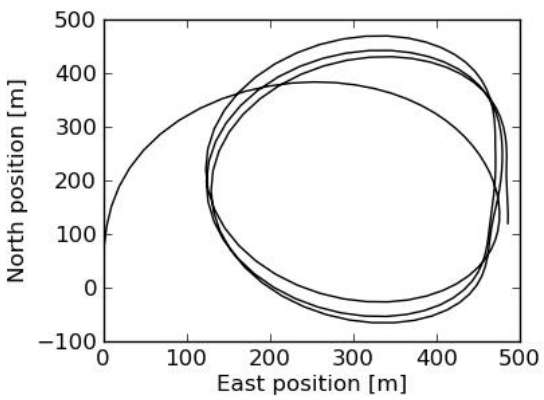


(e) Roll control input.

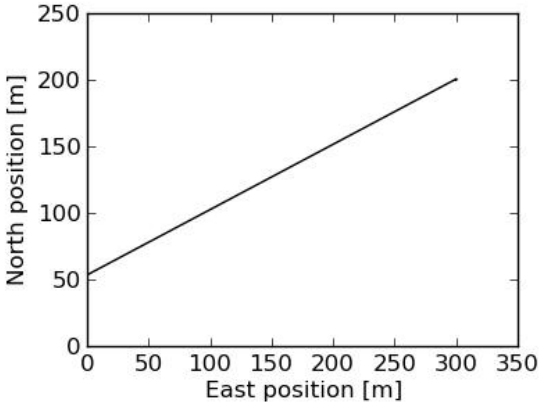
Figure 7.9.: CM1 angles and control variable with wind disturbance.

CM3

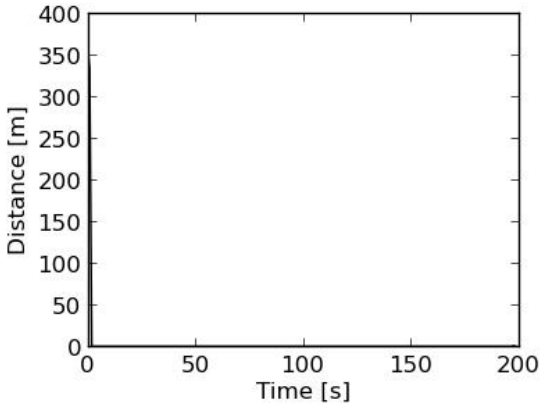
Figure 7.10 and 7.11 shows the simulation with wind with CM3.



(a) East-north map.

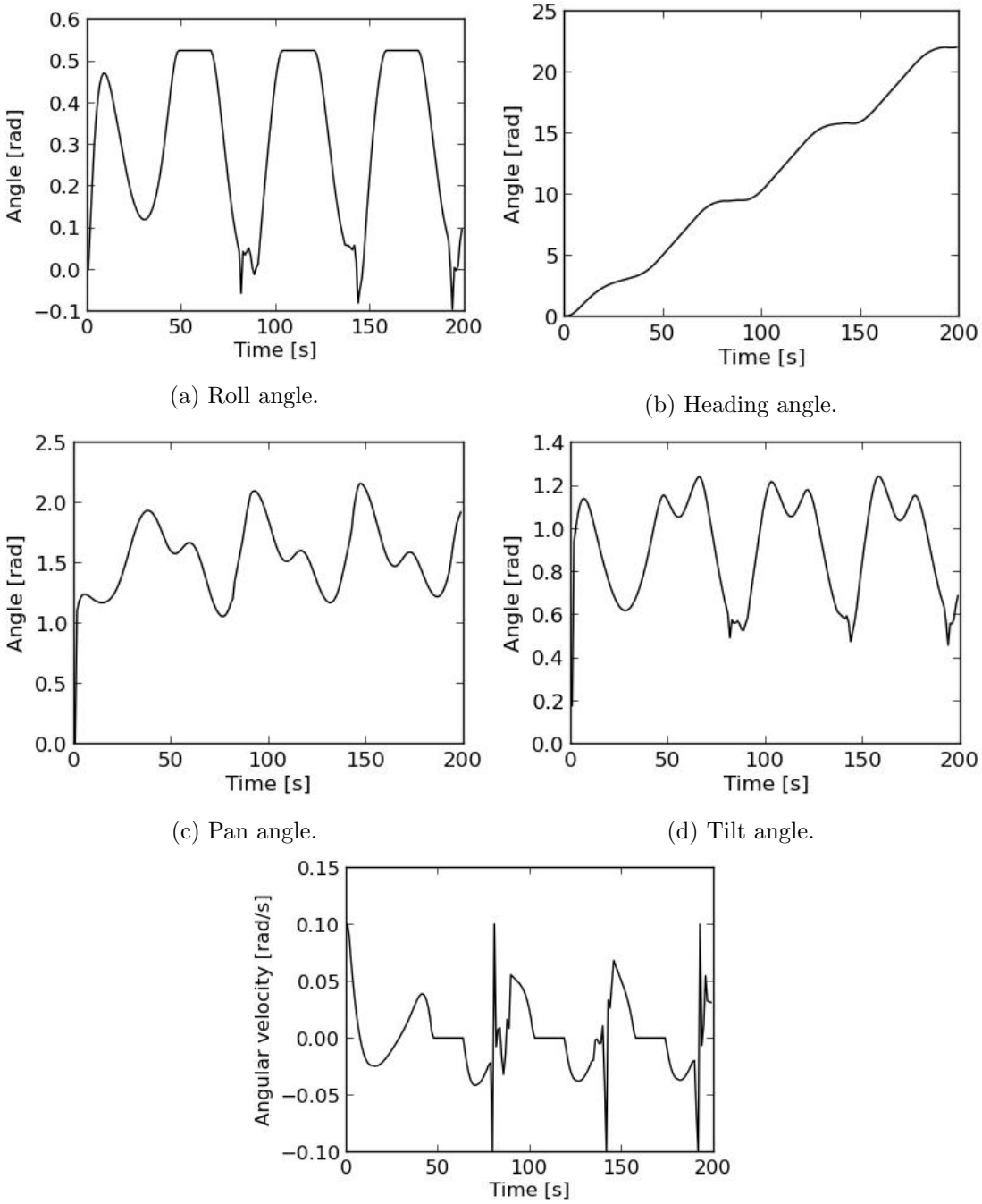


(b) CGP map.



(c) Difference between CGP and object position.

Figure 7.10.: CM3 behaviour with wind disturbance.



(a) Roll angle.

(b) Heading angle.

(c) Pan angle.

(d) Tilt angle.

(e) Roll control input.

Figure 7.11.: CM3 angles and control variable with wind disturbance.

7.2.3. Time Usage

The time usage of the controllers is defined as the time the controller uses to find a new control step.

The time usage of the control methods on the PandaBoard is essential, in order to know whether they can be utilized on board the UAV. Simulations were run on the PandaBoard for CM1 and CM3, and the results can be seen in figure 7.12 and 7.13, respectively. The roll and roll rate for these simulations are also shown, and the relationship between them are discussed in Chapter 9. The scenario simulated here is identical to that of Section 7.2.2, and the results are show to be identical as well.

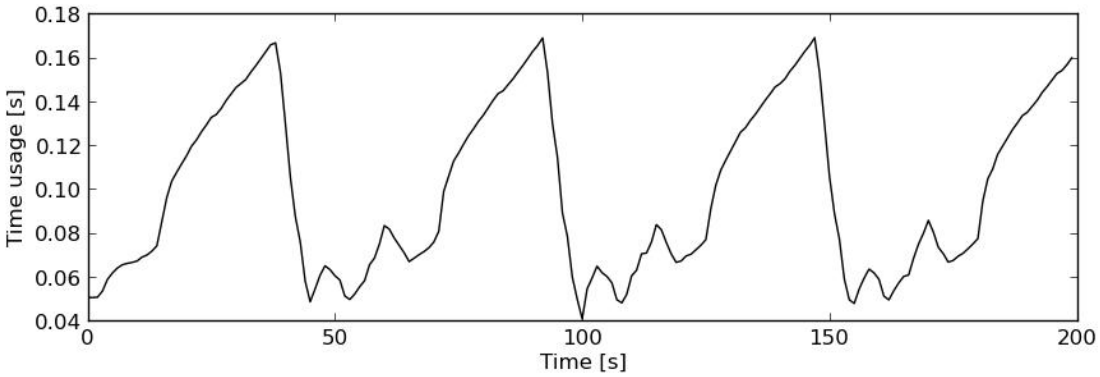


Figure 7.12.: CM1 time usage during simulation on PandaBoard.

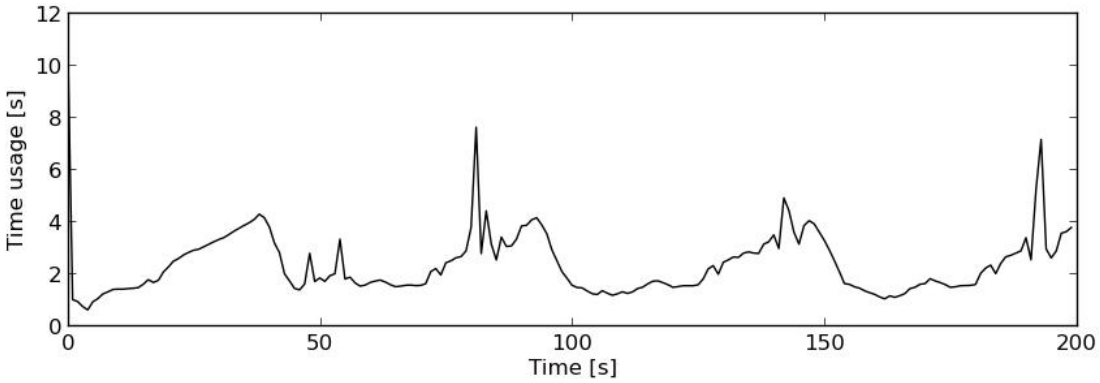


Figure 7.13.: CM3 time usage during simulation on PandaBoard.

8. Hardware-In-the-Loop

8.1. HIL Setup

The hardware-in-the-loop setup requires 12V power supply powering the payload and the Piccolo SL unit, and a PC running the Piccolo Command Center. The PandaBoard is connected to the Piccolo SL unit using a RS232 to USB converter. Because of this, the DUNE program on the PandaBoard needs to be configured to listen for serial communication on the USB port. In the Piccolo Command Center, a configure file specifying that the Piccolo SL unit is mounted backwards on the UAV needs to be loaded. Now all that needs to be done is to start the HIL module from the Piccolo Command Center and to run the DUNE program from the PandaBoard.

Since the Piccolo autopilot does not accept roll rate as control input, a roll value has to be used. Therefore, the roll value one time step into the prediction horizon will be used as the roll control input.

8.1.1. HIL Scenario

The scenario used for HIL testing is that 3 objects on the ground are to be tracked in turn. The objects are not moving, and are distanced some kilometres apart. The UAV is required to be in a proximity of 250m of an object for 60 consecutive seconds before switching target. The locations of the targets are, in GPS coordinates

$$\mu_{obj1} = 0.656088255$$
$$l_{obj1} = -2.134895$$

$$\mu_{obj2} = 0.655894017$$
$$l_{obj2} = -2.13649929$$

$$\mu_{obj3} = 0.65652715308$$
$$l_{obj3} = -2.1360408221$$

which correspond to the position of the three dots in figure D.1.

8.2. HIL Results

8.2.1. Power Consumption

During hours of HIL-testing, the power supply delivering 12V to the system was never observed to draw as much as $2.0A$ current. This corresponds to a maximal use of 24W, and is well below the 60W limit set in 3.3.11.

8.2.2. CM3 HIL Test Not Working

When attempting to HIL test with the CM3 controller on board the PandaBoard, the program suddenly stopped with no warning or error messages. The tests and analysis following this is described in Section 9.2.1.

8.2.3. CM1 HIL Results

Here, the HIL results of the CM1 controller will be presented.

The controller switched target object from object 1 to 2 at time t_1, 2 to 3 at time t_2, 3 to 1 at time t_3, and 1 to 2 at time t_4. These time instances were:

$$t_1 = 581s \tag{8.1a}$$
$$t_2 = 1182s \tag{8.1b}$$
$$t_3 = 1552s \tag{8.1c}$$

In figure 8.1, the path taken by the UAV to track the three objects can be seen.

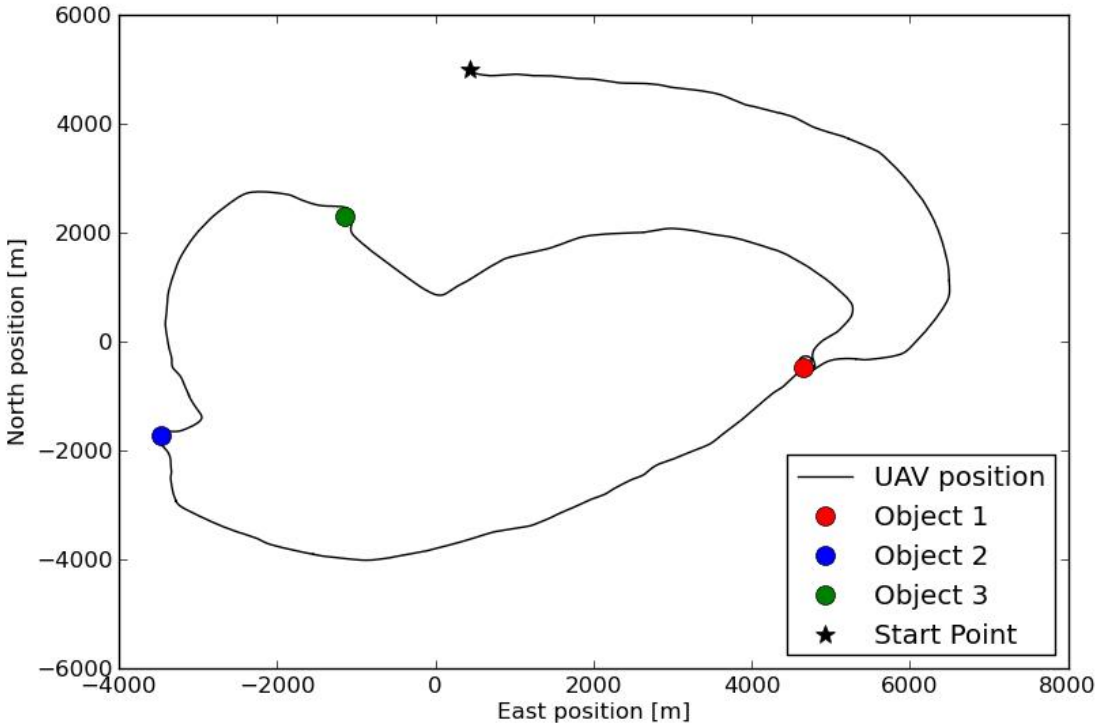


Figure 8.1.: Flight path shown in east-north map.

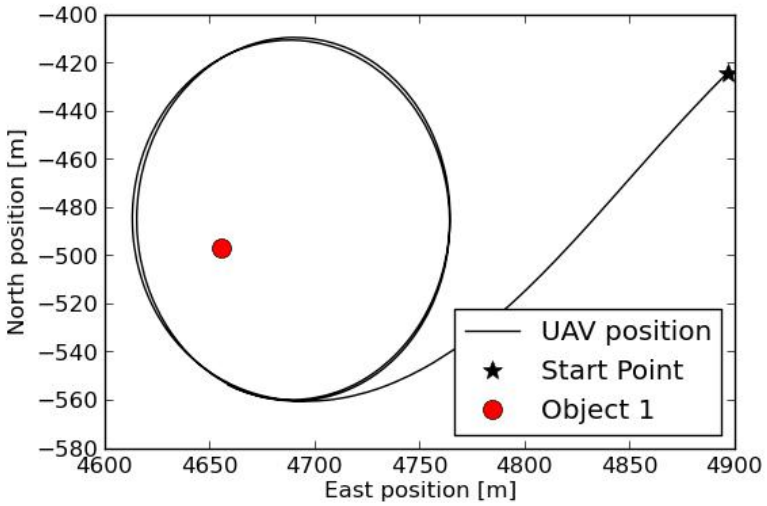
Because the behaviour of the system was bad when far away from the objects, only the behaviour in the 60 last seconds before switching target object is shown. These can be seen in 8.2, 8.4, and 8.6 for object 1, 2, and 3, respectively.

In order to find an estimate of where the CGP must point in order to have the object within the field of view (FOV), let us pretend that the UAV hovers above the object at $h = 117$m, which was the height in the HIL test. Now, the horizontal and vertical angles of the FLIR Tau IR camera is 32° and 26°, respectively. Using the angle of 26° yields a ground distance r between the points in the top and the middle of the camera frame of

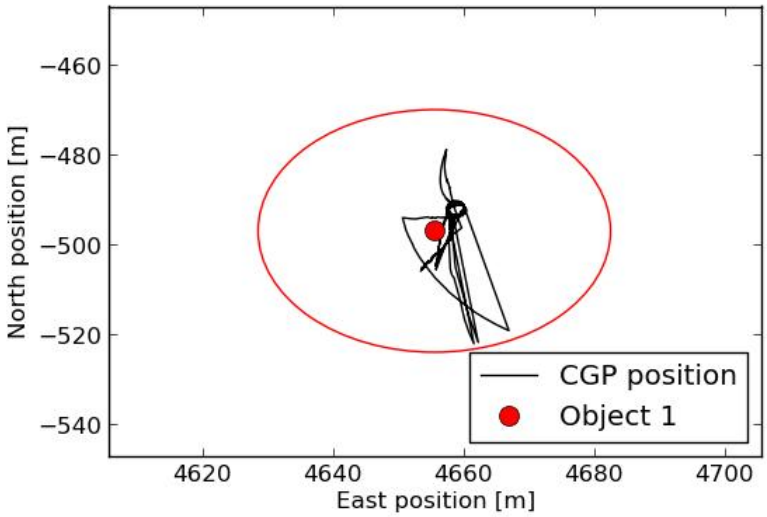
$$r = h \tan\left(\frac{26^\circ}{2}\right) \approx 27m \tag{8.2}$$

Drawing a circle with radius 27m and with center on the object gives us an estimate of where the CGP must lie in order to keep the object in the field of view. This distance can be seen in 8.2b, 8.4b, and 8.6b as a red circle and in 8.2c, 8.4c, and 8.2c as a red line. This line is a pessimistic approximation, as it is the distance from the UAV to the

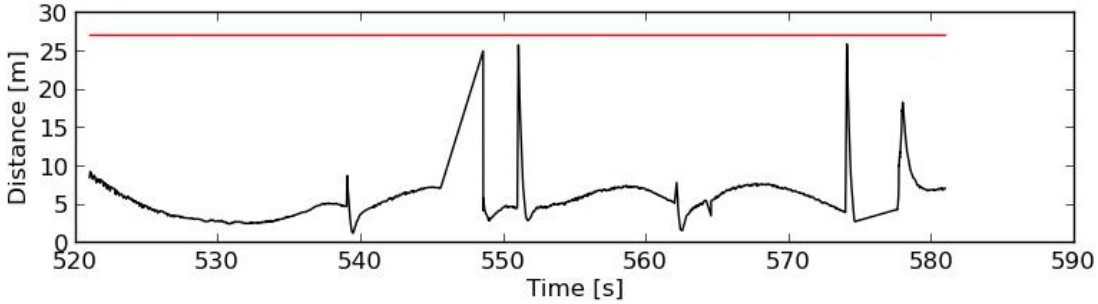
CGP that matters, and not the height over ground. The hight over ground is used since it is the smallest possible value.



(a) Flight path near object 1 shown in east-north map.

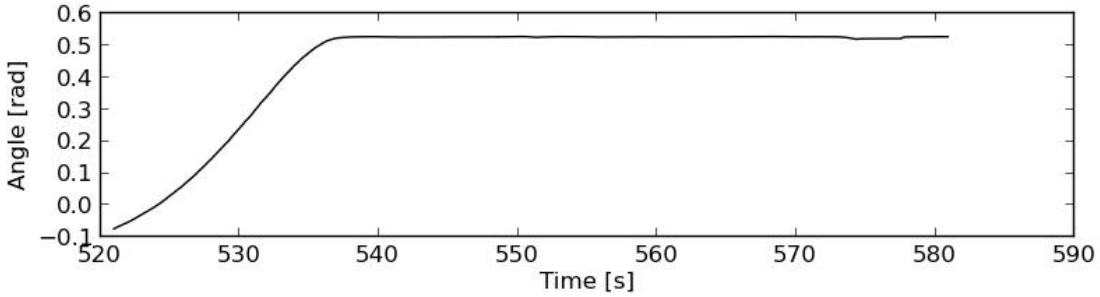


(b) CGP shown in east-north map when tracking object 1.

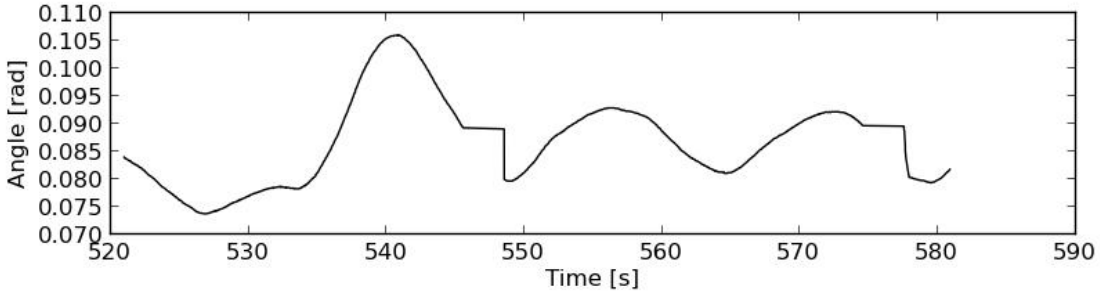


(c) Distance between CGP and object 1.

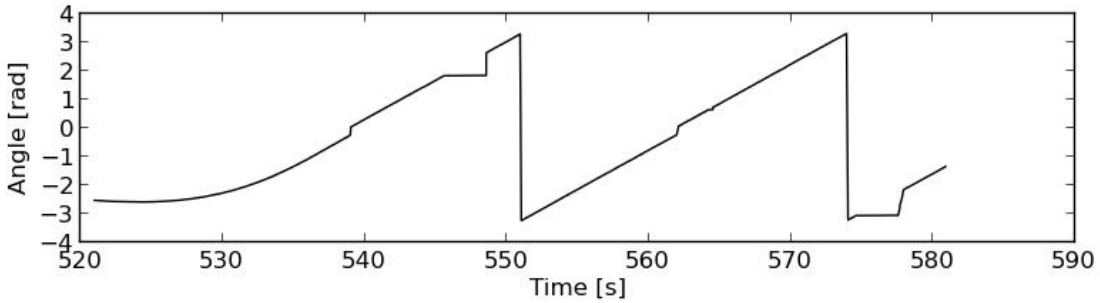
Figure 8.2.: Behaviour near object 1.



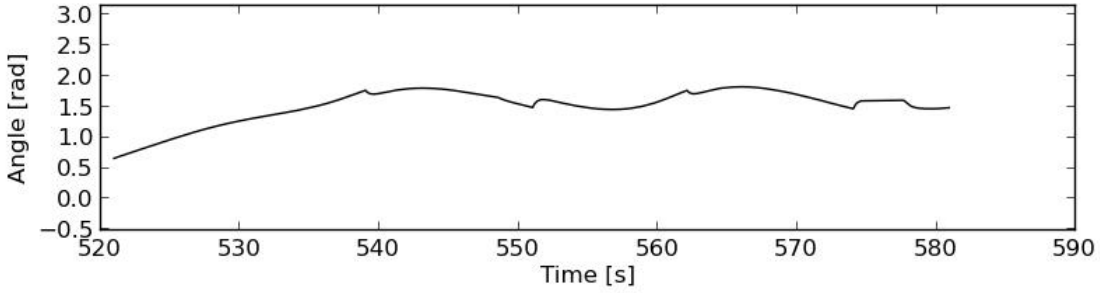
(a) Roll angle.



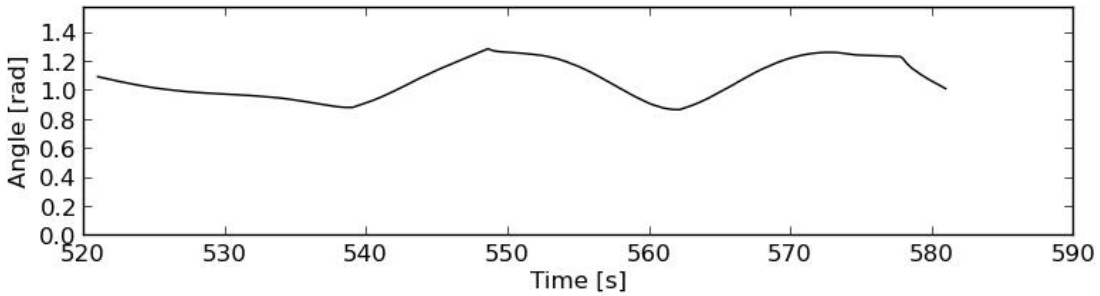
(b) Pitch angle.



(c) Heading angle.

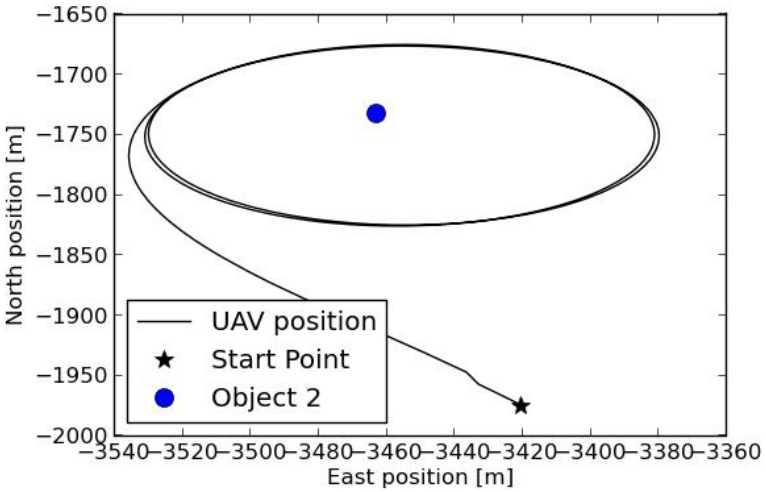


(d) Pan angle.

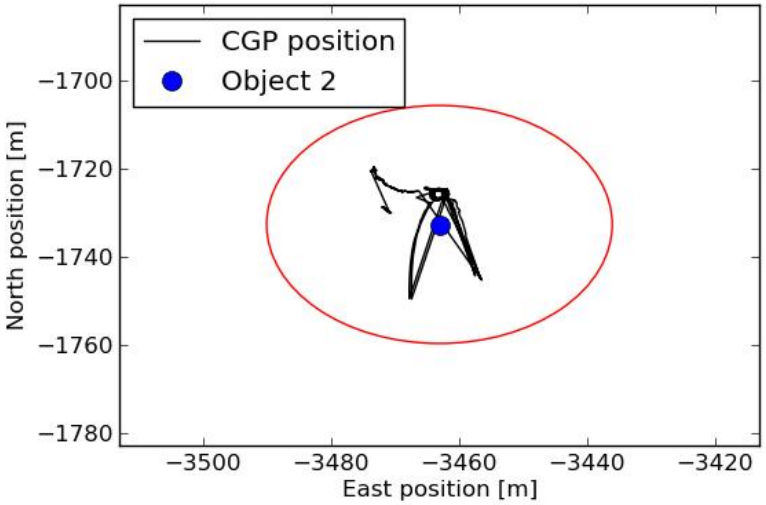


(e) Tilt angle.

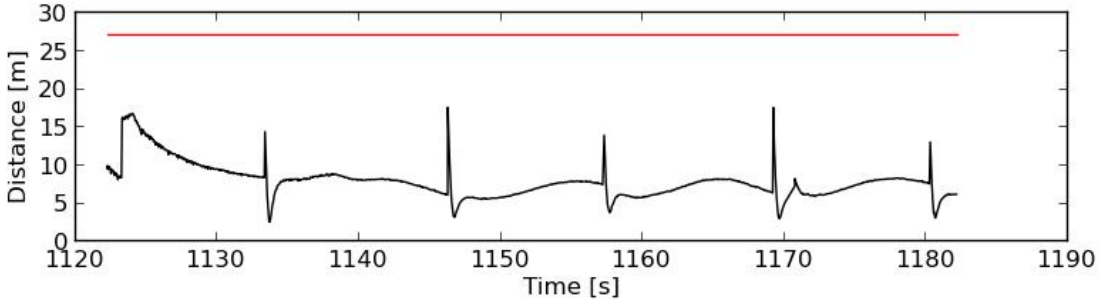
Figure 8.3.: UAV and PTG angles near object 1.



(a) Flight path near object 2 shown in east-north map.

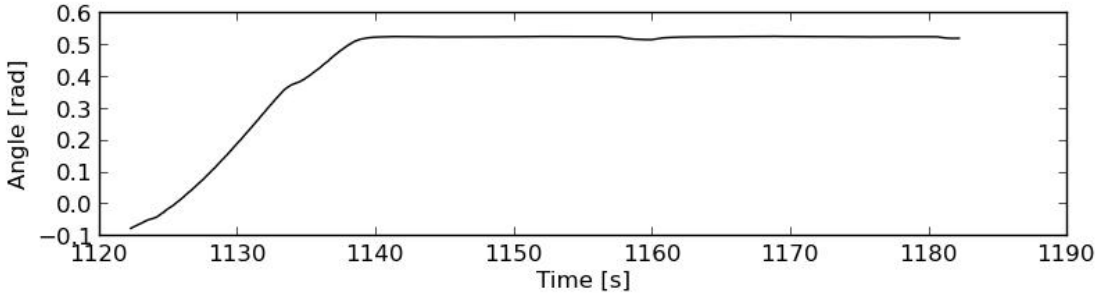


(b) CGP shown in east-north map when tracking object 2.

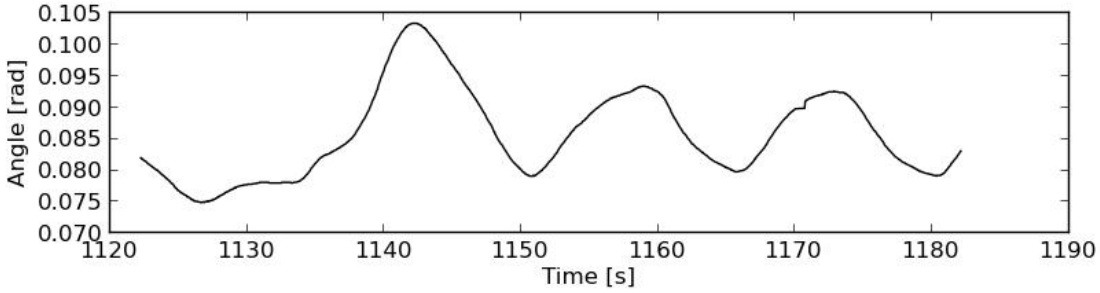


(c) Distance between CGP and object 2.

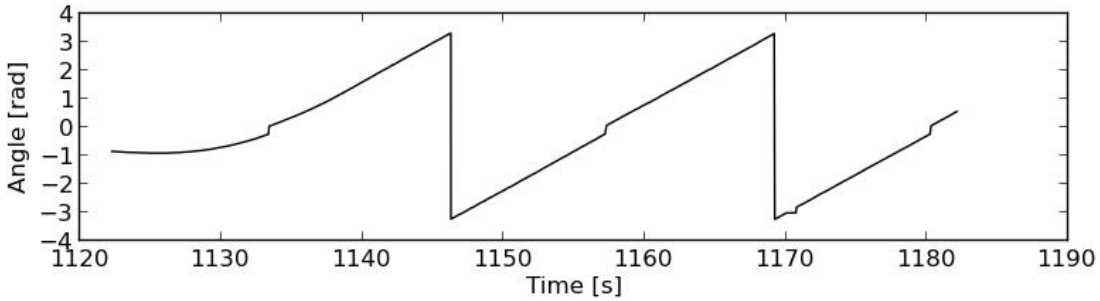
Figure 8.4.: Behaviour near object 2.



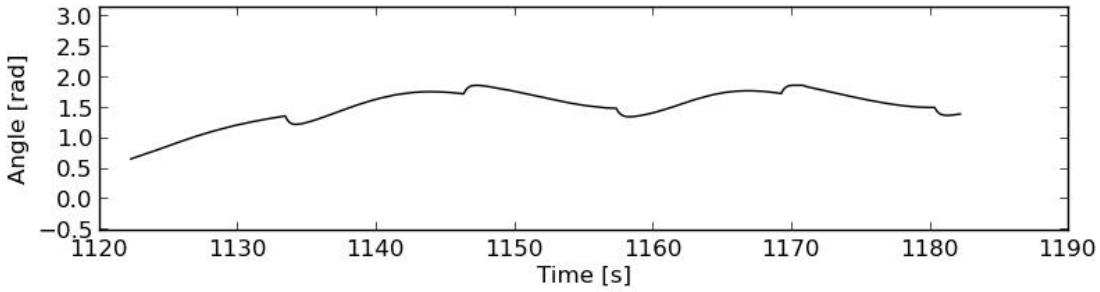
(a) Roll angle.



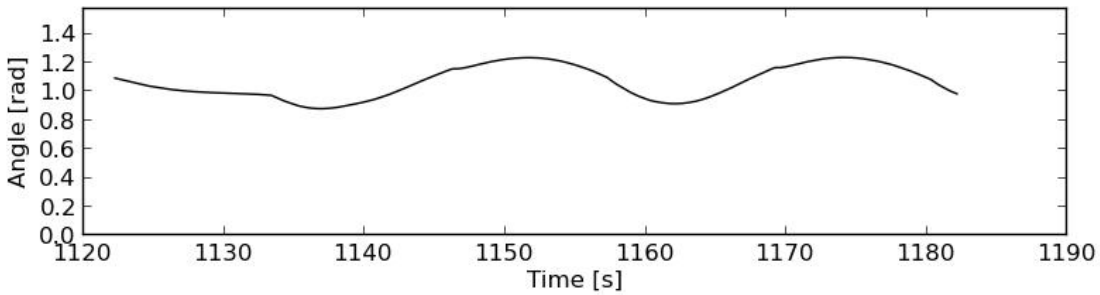
(b) Pitch angle.



(c) Heading angle.

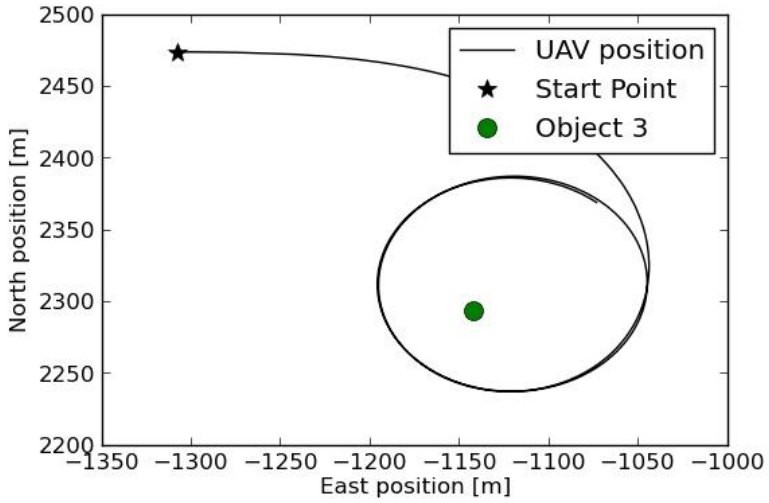


(d) Pan angle.

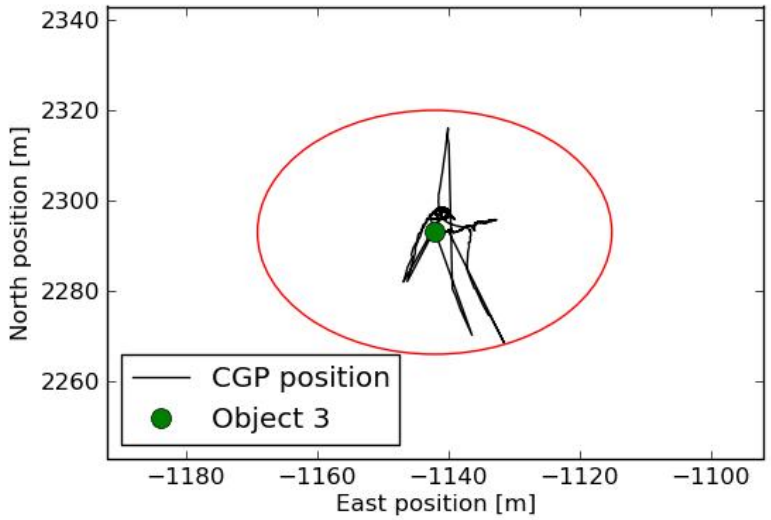


(e) Tilt angle.

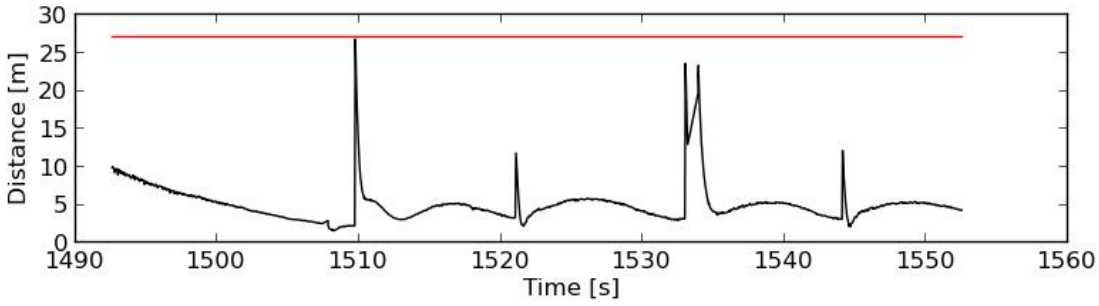
Figure 8.5.: UAV and PTG angles near object 2.



(a) Flight path near object 3 shown in east-north map.

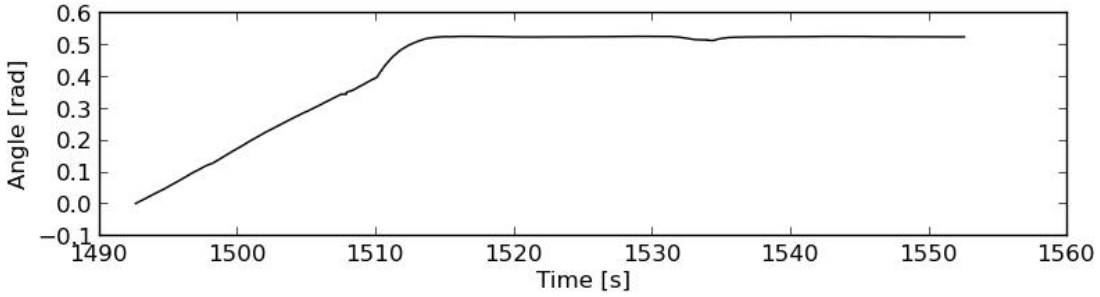


(b) CGP shown in east-north map when tracking object 3.

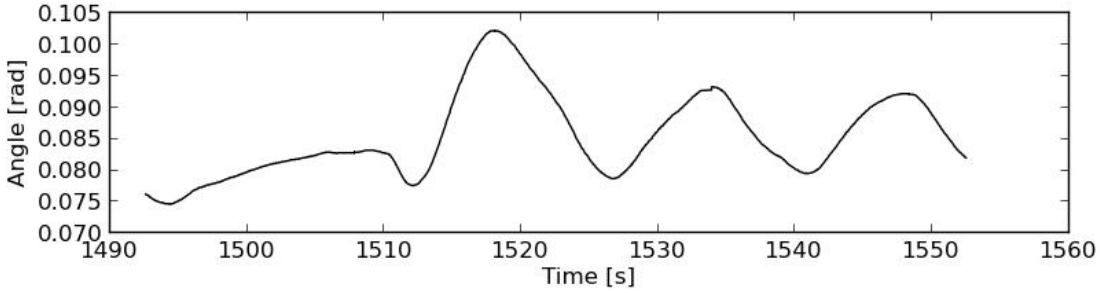


(c) Distance between CGP and object 3.

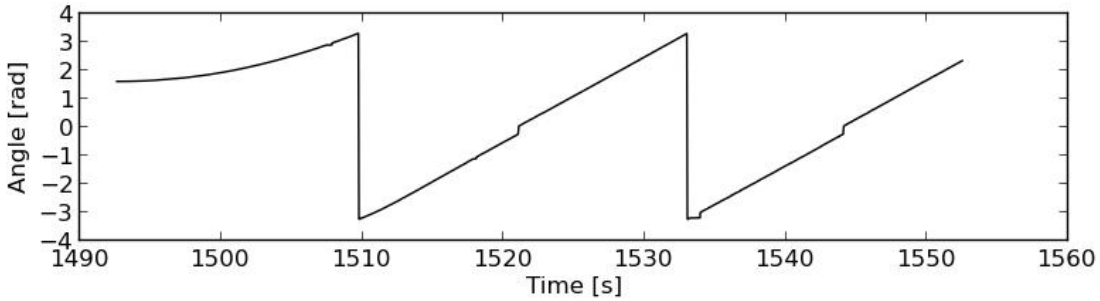
Figure 8.6.: Behaviour near object 3.



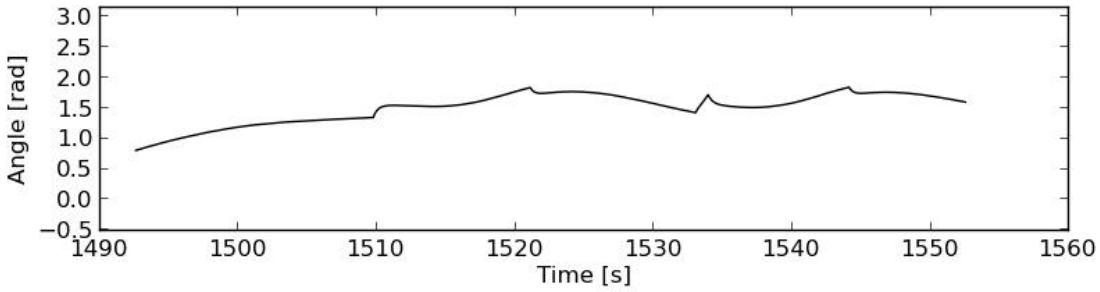
(a) Roll angle.



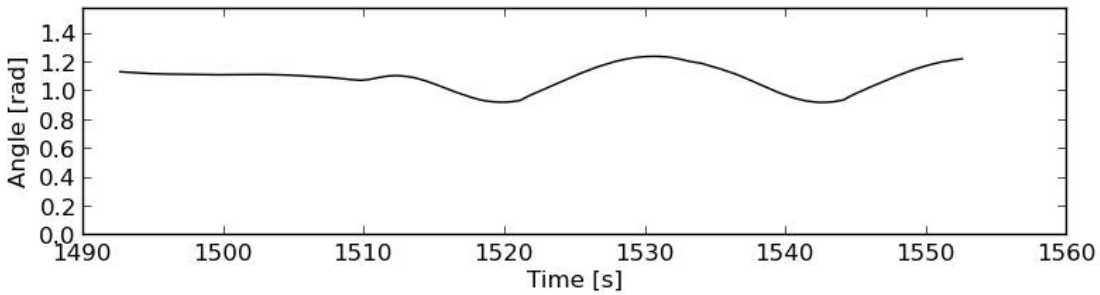
(b) Pitch angle.



(c) Heading angle.



(d) Pan angle.



(e) Tilt angle.

Figure 8.7.: UAV and PTG angles near object 3.

9. Discussion

In this chapter, the results of Chapter 7 and 8 will be discussed. Additionally, thoughts on how to make further improvements on the controllers in the future will be shared.

9.1. Simulations

9.1.1. Optimization Fault in CM1

As mentioned in Section 7.2.1, figure 7.1 clearly shows that something is faulty. Based on extensive testing and search for explanations to this problem, it seems that the optimization routine attempts to minimize the heading's absolute value. This can be seen in figure 7.2b, where the heading is oscillating about 0°, which leads to the 8-shaped flight pattern. With this explanation, a simple solution should be to start with an offset on the heading that is a multiple of 2π as this will have no effect on the model. Adding the starting offset to the initial heading yields a new initial heading:

$$\psi_0' = \psi_0 - 50\pi \tag{9.1}$$

The result of using ψ_0' as initial value can be seen in 7.3 and 7.4 which are the expected results. The unfortunate consequence of this fix is that the turning direction of the circle is predetermined, and is not up to the MPC to determine. This is because a large positive offset in the heading will make the MPC find roll values such that the heading value is decreasing. So by setting the large offset positive, the UAV will primarily roll left and take left turns. Oppositely, by setting the large offset negative, the UAV will primarily take right turns. Evidence of this can be seen in figure B.3 and B.4. Another consequence of this error would be that by setting the offset so small that the heading reaches 0 within the 200 seconds of simulation, the UAV should change from a circular

pattern to an 8-shaped pattern when that happens. This is shown in B.1 and B.2 where the starting offset is -4π.

It is hard to imagine what may cause this faulty behaviour. Adding an offset of $2k\pi$, where k is any integer, to the model yields

$$\begin{aligned}
\dot{x} &= V\cos(\psi + 2k\pi) + V_{w,N} = V\cos(\psi) + V_{w,N} \\
\dot{y} &= V\sin(\psi + 2k\pi) + V_{w,E} = V\sin(\psi) + V_{w,E} \\
\dot{\phi} &= u_\phi \\
\dot{\psi} &= \frac{\mathrm{d}}{\mathrm{d}t}(2k\pi) + \frac{g}{V}\phi = \frac{g}{V}\phi
\end{aligned}$$

which clearly is the same model as the original. Had this problem originated from a programming error, then the same faulty behaviour should be seen both with and without the heading offset. This suggests that there is bug in ACADO that causes this behaviour.

9.1.2. CM2 Simulation Failing to Execute

The simulation of the original CM2 formulation failed to run as mentioned in 7.2.1. The modified CM2 controller attempts to control the PTG to point straight down. This is chosen in order to avoid the non-linearity introduced by dividing by the length of the vector from the UAV to the object. This can be seen in (5.6). The vector $[\bar{x}^n_{o/b}, \bar{y}^n_{o/b}, \bar{z}^n_{o/b}]$ in the cost function (5.9a) is replaced by the vector $[0, 0, 1]$. In figure 7.5c the distance between the object and the UAV in the 2D east-north plane is shown. From this we see that the modified control method manages to point the camera straight down as the distance rapidly goes to zero. This is not a useful result other than for showing that this control method is able to point the camera in a desired direction. The problem might either be caused by a programming error or that ACADO can not handle the added non-linearity in the original formulation. The only principal difference between the two implementations is the vector $\bar{\boldsymbol{p}}^n_{o/b}$, which is very easy to implement in ACADO. This might suggest that the error does not lie with the programmer, but that the formulation is too complex for the ACADO code generation tool.

Because of this, CM2 is neither simulated nor HIL tested any further. When comparing the implementation of CM2 and CM3, the CM2 controller requires more states and has more complexity, which suggests that it would perform worse than the CM3 controller.

9.1.3. Performance Without Wind

The behaviour of both the CM1 and CM3 controllers is satisfying in the case with no wind, as shown in Section 7.2.1. They both circle the object, which can be seen in figure 7.3a and 7.6a, and keep the camera steadily focused on the object, which can be seen in figure 7.3b and 7.6b. This is as expected because there is no wind disturbances or errors between the controller model and simulation model. These plots are only suitable for showing that the CM1 and CM3 controllers are able to control the UAV and camera appropriately.

9.1.4. Performance With Wind

The main source of disturbance in the system is wind. Fortunately, the Piccolo SL unit outputs a wind estimate that can be included in the model. A scenario with 10m/s south wind is simulated and the results are shown in Section 7.2.2. The paths found by both controllers, in figure 7.8a and 7.10a, is seen to be egg-shaped. This is because the when flying southwards, there is a $10\frac{m}{s}$ head wind, causing the ground speed to be reduced by $10\frac{m}{s}$. Oppositely, when flying northwards, the ground speed is increased by $10\frac{m}{s}$. Because of this, the UAV has a lot more time for turning in the eastern part of the path than in the western part. Since the path is optimized with respect to the distance from the UAV to the object, the controllers find it better to straighten the path than keep turning. The only reason that the path does not buckle inwards towards the object on the eastern side of the path is the strong penalty on the roll rate. This is apparent from figure B.5a and B.7a, which shows the same scenario with a penalty on roll rate of 10^6 instead of 10^7.

Comparing the behaviour of the two controllers in the presence of wind in figures 7.8b and 7.10b, the CM3 controller can be seen to show much better behaviour. This is because the presence of wind makes the UAV change its roll angle a lot more, which the CM3 controller compensates for proactively with the pan and tilt angles. The CM1 controller, however, only updates the pan and tilt angles after the effect of the roll change has been measured. This is one of the key differences between the two controllers.

9.1.5. Time Usage

The amount of time the MPC algorithm uses to find the next control step is important for determining whether or not the algorithm can be used on board the UAV. Therefore, a simulation was run on the PandaBoard, and the time usage for each step was logged. The results can be seen in figure 7.12 for CM1 and 7.13 for CM3.

CM1 can in figure 7.12 be seen to maximally use less than 0.2 seconds to find a control step. This corresponds to 5 times per second, which should be more than enough to be used on board the UAV. However, this is a nearly ideal case, where the starting point is near the object and the wind is known exactly during the whole simulation. Therefore, the frequency with which the MPC algorithm can be run on board the UAV in the HIL testing might not be so high. This result still gives hope that the solution gives good control in the HIL testing.

The performance of the CM3 controller on the PandaBoard, seen in figure 7.13, is much slower. It uses at most about 8 seconds, and on average over 2 seconds. Taking the near ideal conditions of this simulation into account, the controller can be assumed to run even slower in an actual flight scenario, especially when farther away from the object. Since this controller controls the PTG, the camera control becomes very vulnerable to model errors, as it may take over 2 seconds before they are corrected.

The poor speed performance of the CM3 controller is caused both by the complexity the of the formulation and the slow computer hardware. The PandaBoard is not designed for high performance purposes, and much faster computer hardware might enable the use of this controller. In figure B.9, the performance of the CM3 controller is shown when the same simulation is run on a laptop PC. Now it is seen to use maximally 0.4 seconds to find a control step. This means that better computer hardware must be available for this to be an applicable controller. It is not unrealistic to get better, specialized computer hardware that meets the performance requirements and is small enough to fit on board the UAV. Additionally, as seen in 8.2.1, there is enough power for a more power demanding unit as well.

9.2. HIL simulations

9.2.1. CM3 HIL Test Failing to Execute

The HIL test with the CM3 controller stopped without any warnings or error messages being printed. First, the initial values that the MPC was about to use right before the crash were gathered and used in a simulation. The simulation ran successfully, ruling out that the initial values led to feasibility issues and made ACADO crash. The place where the program crashed was narrowed down to be right before the ACADO function feedbackStep(), a function in which vectors containing thousands of elements are calculated on. This is the function where the heavy MPC computation is performed, which makes memory and CPU usage interesting to check for. DUNE has a task called "Monitors.CPU" which monitors CPU usage and outputs an error message when a a certain threshold is met. Neither altering the threshold nor removing the task from the system had any impact at all. Exactly the same system was run on a laptop computer with much better hardware specifications, but the program still crashed without any indication of what was wrong. The CM1 controller which successfully ran the HIL test was altered in order to be more computationally heavy. The prediction horizon was simply set to 100 seconds, which makes the vectors handled in feedbackStep() much larger. Now, the same behaviour as with the CM3 controller was experienced. This suggests that DUNE either has a problem dealing with the complexity of the ACADO program or that DUNE has some limit other than CPU usage that the MPC violates and fails to print an error message. The latter may e.g. be violation of a memory usage limit.

This problem was not successfully resolved, which makes the CM3 controller inapplicable to HIL testing.

9.2.2. CM1 Hil Test

The results of the HIL test can be seen in the figures in Section 8.2. Figure 8.1 shows that the CM1 controller manages to control the UAV to track the objects. The path that the UAV travels, however, is far from optimal. Optimally, nearly straight lines between the objects should be seen. Instead, a much longer, curvy path is taken. Several of the assumptions and simplifications that are made may have an impact on the control of the UAV. Since the behaviour near the objects are much better, as seen in figure

8.2a, 8.4a, and 8.6a, these assumptions and simplifications are not considered to be the primary cause. By the time instances at which the targets were switched, seen in equation 8.1, we see that travelling from one object to another took 6-10 minutes. As discussed in Chapter 5, the prediction horizon was set to 35 seconds in order to capture one whole round of circling the object. Therefore, the MPC is not designed for ensuring good behaviour when far away. Additionally, the Gauss Newton Hessian approximation, mentioned in 2.4.1, is much more inaccurate when the UAV is far away from a local minimum, which is the case when far away from the object. These are considered to be the two primary reasons for this suboptimal behaviour.

In the case where the UAV is far away from the object, a better solution than the MPC in this thesis would be to feed the autopilot a desired heading set point. Then the MPC controllers found here could be turned on when approaching the objects for smoother tracking and gimbal control.

Looking at the CGP position relative to the targeted object in figure 8.2b, 8.4b, and 8.6b, we can see that the CGP stays near the objects. In 8.2c, 8.4c, and 8.6c, the distances are shown. The red line signifies a line on the ground that is 27m away from the object, as discussed in Section 8.2.3, and any CGP inside this line is assumed to capture the object in the field of view. The CGP can be seen to stay within the 27m radius of the object almost all the time which is satisfactory behaviour.

When recording video, it is also important that the camera does not move too quickly, in order for the image not to become blurry. The spikes seen in figure 8.2c and 8.6c therefore show a dissatisfying behaviour of the camera control. Investigating the spikes further, it was found that they may not all be caused by the controller. In the logfile, there was an interval where no estimated states messages were received between $t = 454s$ and $t = 458s$, which can be seen as flat lines in the pitch and heading plots in figure 8.3b and 8.3c. There is also a gap between $t = 574s$ and $t = 577$, which can be seen in the same figures. Therefore, the spike around $t = 456s$ is caused by the fact that new state messages are not received. A solution to this could be to generate state estimates between measurements, which maybe should be implemented anyway in case of longer communication breakdowns.

The other spikes, seen near object 1 and object 3, seem to occur simultaneously as the heading angle reaches π and is set to $-\pi$. In the log file at $t = 551s$, the heading angle goes from 3.2601 to -3.2714 which clearly is not correct. The fault here is not that the heading measurements lie outside $[-\pi, \pi]$, but that a value of $\pi + \delta$, where δ is the

small deviation from $\pm\pi$, should be set to $-\pi + \delta$ and not $-\pi - \delta$, which clearly is done here. Because of this, all sinus operations with heading gets the wrong sign at these time instances, which is assumed to be the sole reason for these spikes. To clarify, letting the heading angle take values beyond $[-\pi, \pi]$ does actually makes sense. In case the UAV travels directly southwards, one could get a heading estimate that fluctuates rapidly from $\approx -\pi$ to $\approx \pi$. Requiring that the heading must be $\pi + \delta$ before setting it to $-\pi + \delta$ fixes this.

When considering that these spikes seem to be caused either by a fault in an other project or a small communication breakdown that has the simple fix described above, the camera control results must be said to be satisfactory.

In the HIL test shown in Section 8.2, the UAV's pitch angle was assumed to be zero when finding the CGP. As seen in figure 8.3b, 8.5b, and 8.7b this was not the case. Therefore, a second HIL test was performed, where the pitch angle was taken into account when finding the CGP, but neglected when finding the desired pan and tilt angles. This way, something can be said about the effect of neglecting pitch angle.

In Appendix C, the results of calculating the CGP point with the true pitch value is shown. The starting point is different from the first HIL simulation which makes the plots a little different, but the performance can be seen to be nearly as good. When comparing the two HIL simulation results, the effect of assuming that the pitch is zero can be seen to be small. The distance between the object and the CGP is a little longer with the pitch error, but the CGP is still within the 27m radius almost all the time. Therefore, neglecting the pitch angle is an appropriate simplification, at least when it is as small as $0.09 \approx 5°$.

9.3. Assessment of the Controllers

A better approach for the CM1 controller would be to find the desired pan and tilt angles by taking the pitch angle into account. Then, the performance would probably be similar to the HIL results in Section 8.2. The only alteration needed for this, is to change the rotation matrix $R_n^b(\phi, \psi)$ in equation (5.3) with the rotation matrix $R_n^b(\phi, \theta, \psi)$ found in e.g. Fossen [2011].

Knowing that neglecting the pitch has little influence on the performance is a useful result for the CM2 and CM3 controllers. In the simulations, the CM3 controller was

seen to achieve better camera control than CM1 because it compensated for an upcoming change in roll with the pan and tilt angles. On this background, the CM3 controller is assumed to achieve better camera control than CM1 if better hardware should become available on the UAV, allowing the CM3 controller to run fast enough.

Because the predicted future roll value is available to the CM1 controller through the MPC solution the pan and tilt angles could be found using it. With proper tuning of such a solution, this could negate the advantage of the CM3 controller over the CM1 controller, without the need for better computer hardware. Therefore, in the future it seems advisable to pursue this solution instead of trying to enhance the hardware and get the CM3 controller to run in DUNE.

Another point that makes the CM3 solution inadvisable for future work is that the way it couples PTG and UAV control is only useful when the flight behaviour hinders the camera from catching the object in the field of view. This could e.g. be true if there are strong limitation on pan and tilt, or if the gimbal is of the type tilt or fixed. Then, knowing that the camera will be able to film the object in the duration of the prediction horizon is useful information. If the UAV flight pattern when near the object never hinders the camera from filming it, however, this information is useless. Therefore, instead of the CM3 controller as implemented here, a new control method is proposed. By controlling the UAV and the PTG separately with their own MPC controller, the predicted future states from the UAV controller could be sent to the PTG controller. Now, the PTG controller could use this information to find pan and tilt angle controls that minimizes the distance between the CGP and the object over a prediction horizon of 2-5 seconds. The UAV controller is already shown to work, and this would greatly reduce the complexity of the PTG control compared to the CM3 controller. This proposed solution is also preferred because it could possibly be implemented and run on the PandaBoard or some other inexpensive small unit.

9.4. Assessment of ACADO

The ACADO toolkit has been a vital part of this thesis. It has proven to be a powerful and user-friendly toolkit. It is still in its beta phase, however, and a lot of its functionality is poorly documented. During the course of this spring, a lot of time has been spent trying to make ACADO work, and a few times, the solution has been a bug fix from

the ACADO developers. However, through the ACADO community website[1], both the developers and users have been helpful and friendly. All in all, ACADO is strongly recommended for future use in the field of UAV control.

[1] http://sourceforge.net/p/acado/wiki/Home/

10. Conclusion

In the work that lies behind this thesis, the payload hardware shown in figure E.1 was built and software needed to control the UAV was developed in collaboration with M.Sc. Stian Nundal, Espen Skjong, and Carl Magnus Mathisen and PhD candidate Frederik Leira. In this thesis, a robot manipulator model of the UAV and PTG system was developed, which leaves an extendable framework for future use. Through this, the flight dynamics and, most importantly, the relationship between the UAV and PTG angles and the ground point at which the camera points was found. Further, the use of the MPC control methodology in UAV search and rescue missions were explored. Three principles of controlling the UAV and a pan-tilt gimbal were developed and the corresponding controllers were implemented in ACADO. The two most complex controllers had problems running and were not HIL tested. One of them, however, was successfully simulated, and showed promising performances. The simplest controller was successfully implemented in DUNE and HIL tested, and yielded excellent results. In figure 8.2b, 8.4b, and 8.6b, and figure C.2c, C.4c, and C.6c, the object was shown to remain in the field of view at all times during the 60 seconds of tracking. The spikes in the plots that left the object outside the field of view were shown to be caused by either errors in other projects that this thesis depends on, or a short communication breakdown that needs to be fixed elsewhere than in the controller. Additionally, the red line in the plots that approximated where the camera must point in order to keep the object in the field of view, was shown to be a pessimistic approximation. This means that the margin of error is even bigger than shown in the plots.

For future work, a hybrid of two of the controllers was recommended. In this solution, a UAV MPC controller, that is similar to the simplest controller in this thesis, is used. This sends the predicted future states of the UAV to a MPC PTG controller which has a much shorter prediction horizon, and is based on the same equations as the complex controllers of this thesis. This solution would probably even be able to run fast enough on a PandaBoard, relieving the need for better, more expensive hardware. It should be

noted that when far away from the controller, a simple line of sight controller would be advised as the MPC algorithm was neither designed nor suitable for this purpose.

The results of this thesis are promising for use in the real world. With small modifications, both the controller found in this thesis and the proposed controller for future research can achieve good autonomous UAV and PTG control on board a UAV with limited computing power resources.

Appendix

A. ACADO

A.1. Functionality

ACADO allows for simple and intuitive problem formulation and solution with a lot of optimization methods. Here, a small review of the functionality used in this thesis is presented. A very simple control problem is used to showcase the functionality of ACADO, and to show the implementation of slack variables mentioned in Section 2.3.1.

The user is initially required to specify the state variables, control variables and differential equations of the system:

```
DifferentialState x, d; \\d is a dummy state needed for the slack variable s
Control u, s; \\s it the slack variable
DifferentialEquation f;
f << dot(x) == x*x+u;
f << dot(d) == s;
```

An optimal control object can now be created and the least squares function defined:

```
double t_start = 0.0;
double t_end = 10.0;  \\Horizon length
int N = 20; \\Number of control steps
OCP ocp(t_start, t_end, N);
Function h;
h << x << s;
Matrix Q(2,2);
Q(0,0) = 1;
Q(1,1) = 10;
ocp.minimizeLSQ(Q,h);
```

The mathematical model and constraints of the system can now be implemented:

```
ocp.subjectTo(f);
ocp.subjectTo(u_min <= u + s <= u_max);
```

The code export facility can now be created, export options set and the code exported to the folder "export":

```
OCPexport mpc;
mpc.set(HESSIAN_APPROXIMATION, GAUSS_NEWTON);
mpc.set(INTEGRATOR_TYPE, INT_RK45);
mpc.set(QP_SOLVER, QP_OASES);
mpc.exportCode("export");
```

Finally, the folder "export" is created, and this includes all the functionality needed for fast NPMC problem solving. By including the header files in the "export" folder, the programmer has access to the following:

- ACADOVariables - a struct in which the initial states x_0 and the reference trajectory $\eta(t)$ can be set, and an optimal control step, u, can be extracted
- void initializeSolver() - a function that needs to be called before any other exported function can be called
- void preparationStep() - a function which prepares for the next NMPC iteration
- void feedbackStep() - performs an NMPC iteration

A.2. CM1

```
int main( ){
USING_NAMESPACE_ACADO
double speed = 28;
double grav = 9.81;

DifferentialState N;
DifferentialState E;
```

```
DifferentialState w_N;
DifferentialState w_E;
DifferentialState phi;
DifferentialState psi;
Control u;

//DEFINE DYNAMIC SYSTEM
DifferentialEquation f;
f << dot(N) == cos(psi)*speed + w_N;
f << dot(E) == sin(psi)*speed + w_E;
f << dot(w_N) == 0;
f << dot(w_E) == 0;
f << dot(phi) == u;
f << dot(psi) == phi*grav/speed;

Function h;
h << N;
h << E;
h << u;
Function hN;
hN << N;
hN << E;

Matrix Q(3,3);
Q.setZero();
Q(0,0) = 1;
Q(1,1) = 1;
Q(2,2) = 1000000;
Matrix QN(2,2);
QN.setZero();
QN(0,0) = 1;
QN(1,1) = 1;

OCP ocp(0, 35, 35);

ocp.minimizeLSQ(Q,h);
```

```
ocp.minimizeLSQEndTerm(QN, hN);

ocp.subjectTo(f);
ocp.subjectTo( -PI*30/180 <= phi <= PI*30/180);
ocp.subjectTo( -0.1 <= u <= 0.1);

OCPexport mpc(ocp);
mpc.set( HESSIAN_APPROXIMATION,GAUSS_NEWTON );
mpc.set( DISCRETIZATION_TYPE,MULTIPLE_SHOOTING );
mpc.set( INTEGRATOR_TYPE,INT_RK78 );
mpc.set( QP_SOLVER,QP_QPOASES );

if (mpc.exportCode( "./export" ) != SUCCESSFUL_RETURN)
exit( EXIT_FAILURE );
return EXIT_SUCCESS;
}
```

A.3. CM2

```
int main( ){
USING_NAMESPACE_ACADO
double speed = 28; //speed
double grav = 9.81;

double t_start = 0.0;
double t_end = 35;
int N_iter = 35;

DifferentialState x;
DifferentialState y;
DifferentialState z;
DifferentialState w_N;
DifferentialState w_E;
DifferentialState phi;
DifferentialState psi;
```

```
DifferentialState alpha;
DifferentialState beta;
DifferentialState d1;
DifferentialState d2;
DifferentialState x_ref;
DifferentialState y_ref;
Control u_phi;
Control u_alpha;
Control u_beta;
Control s1;
Control s2;

////////////////////////////////////
Expression rx = x_ref-x;
Expression ry = y_ref-y;
Expression rz = z;

Expression len2 = rx*rx + ry*ry + rz*rz;
Expression len = len2.getSqrt();

Expression xbar_ref = rx/len2;
Expression ybar_ref = ry/len2;
Expression zbar_ref = rz/len2;

Expression xbar = (-sin(psi)*cos(phi)*sin(alpha)+cos(psi)*cos(alpha))*sin(beta)
+sin(psi)*sin(phi)*cos(beta);
Expression ybar = (cos(psi)*cos(phi)*sin(alpha)+sin(psi)*cos(alpha))*sin(beta)
-cos(psi)*sin(phi)*cos(beta);
Expression zbar = sin(phi)*sin(alpha)*sin(beta)+cos(phi)*cos(beta);
////////////////////////////////////

//DEFINE DYNAMIC SYSTEM
DifferentialEquation f;

f << dot(x) == cos(psi)*speed + w_N;
f << dot(y) == sin(psi)*speed + w_E;
```

```
f << dot(z) == 0;
f << dot(w_N) == 0;
f << dot(w_E) == 0;
f << dot(phi) == u_phi;
f << dot(psi) == phi*grav/speed;
f << dot(alpha) == u_alpha;
f << dot(beta) == u_beta;
f << dot(d1) == s1;
f << dot(d2) == s2;
f << dot(x_ref) == 0.0;
f << dot(y_ref) == 0.0;

Matrix Q(8,8);
Q.setZero();
Q(0,0) = 1;
Q(1,1) = 1;
Q(2,2) = 1e3;
Q(3,3) = 1e3;
Q(4,4) = 1e3;
Q(5,5) = 1e7;
Q(6,6) = 1e9;
Q(7,7) = 1e9;
Function h;
h << x;
h << y;
h << xbar-xbar_ref;
h << ybar-ybar_ref;
h << zbar-zbar_ref;
h << u_phi;
h << s1;
h << s2;

Matrix QN(5,5);
QN(0,0) = 1;
QN(1,1) = 1;
QN(2,2) = 1e3;
```

```
QN(3,3) = 1e3;
QN(4,4) = 1e3;
Function hN;
hN << x;
hN << y;
hN << xbar-xbar_ref;
hN << ybar-ybar_ref;
hN << zbar-zbar_ref;

OCP ocp(t_start, t_end, N_iter);
ocp.minimizeLSQ(Q, h);
ocp.minimizeLSQEndTerm(QN,hN);

ocp.subjectTo(f);

ocp.subjectTo( -30*PI/180 <= phi <= 30*PI/180 );
ocp.subjectTo( -PI <= alpha + s1 <= PI );
ocp.subjectTo( PI*10/180 <= beta + s2 <= PI*90/180 );
ocp.subjectTo( -0.1 <=  u_phi  <= 0.1);
ocp.subjectTo( -PI <= u_alpha <= PI );
ocp.subjectTo( -PI/2 <= u_beta <= PI/2 );

OCPexport mpc(ocp);
mpc.set( HESSIAN_APPROXIMATION, GAUSS_NEWTON );
mpc.set( DISCRETIZATION_TYPE, MULTIPLE_SHOOTING );
mpc.set( INTEGRATOR_TYPE, INT_RK78 );

mpc.set( QP_SOLVER, QP_QPOASES );
mpc.set( GENERATE_TEST_FILE, NO );
mpc.set( GENERATE_MAKE_FILE, NO );
mpc.set( GENERATE_MATLAB_INTERFACE, NO );
mpc.set( GENERATE_SIMULINK_INTERFACE, NO );

if (mpc.exportCode( "./export" ) != SUCCESSFUL_RETURN)
exit( EXIT_FAILURE );
return EXIT_SUCCESS;
```

```
}
```

A.4. CM3

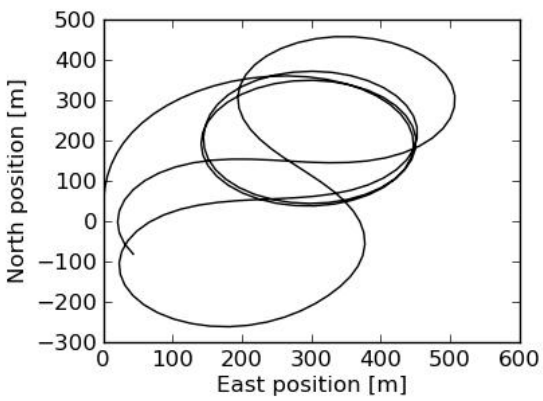
```
int main( ){
USING_NAMESPACE_ACADO
double speed = 28;
double grav = 9.81;
double D_ref = 0.0;

DifferentialState N;
DifferentialState E;
DifferentialState D;
DifferentialState w_N;
DifferentialState w_E;
DifferentialState phi;
DifferentialState psi;
DifferentialState alpha;
DifferentialState beta;
DifferentialState da;
DifferentialState db;

Control u_phi;
Control u_alpha;
Control u_beta;
Control sa;
Control sb;

Expression len = (D_ref-D)/(sin(phi)*sin(alpha)*sin(beta)+cos(phi)*cos(beta));
Expression dN =((-sin(psi)*cos(phi)*sin(alpha)+
            cos(psi)*cos(alpha))*sin(beta)+sin(psi)*sin(phi)*cos(beta))*len;
Expression dE =((cos(psi)*cos(phi)*sin(alpha)+
            sin(psi)*cos(alpha))*sin(beta)-cos(psi)*sin(phi)*cos(beta))*len;

//DEFINE DYNAMIC SYSTEM
```

```
DifferentialEquation f;
f << dot(N) == cos(psi)*speed + w_N;
f << dot(E) == sin(psi)*speed + w_E;
f << dot(D) == 0;
f << dot(w_N) == 0;
f << dot(w_E) == 0;
f << dot(phi) == u_phi;
f << dot(psi) == phi*grav/speed;
f << dot(alpha) == u_alpha;
f << dot(beta) == u_beta;
f << dot(da) == sa;
f << dot(db) == sb;


Matrix Q(7,7);
Q.setZero();
Q(0,0) = 1;
Q(1,1) = 1;
Q(2,2) = 800;
Q(3,3) = 800;
Q(4,4) = 10000000;
Q(5,5) = 1000000000;
Q(6,6) = 1000000000;

Function h;
h << N;
h << E;
h << dN+N;
h << dE+E;
h << u_phi;
h << sa;
h << sb;

Matrix QN(4,4);
QN(0,0) = 1;
QN(1,1) = 1;
```

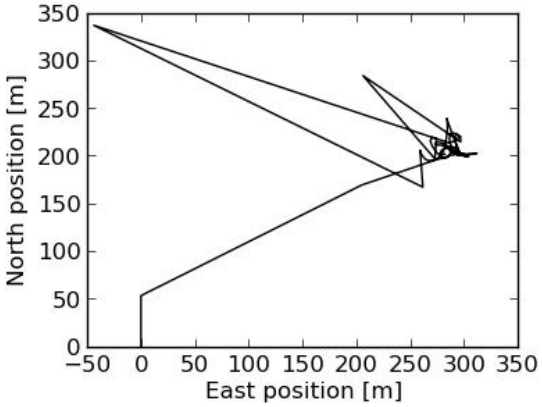
```
QN(2,2) = 800;
QN(3,3) = 800;
Function hN;
hN << N;
hN << E;
hN << dN+N;
hN << dE+E;

OCP ocp(0, 35, 35);
ocp.minimizeLSQ(Q, h);
ocp.minimizeLSQEndTerm(QN,hN);

ocp.subjectTo(f);

ocp.subjectTo( -PI*30/180 <= phi <= PI*30/180 );
ocp.subjectTo( -PI <=  alpha + sa <= PI );
ocp.subjectTo( 0 <= beta + sb <= PI*90/180 );
ocp.subjectTo( -0.1 <=  u_phi  <= 0.1);
ocp.subjectTo( -PI <= u_alpha <= PI );
ocp.subjectTo( -PI/2 <= u_beta <= PI/2 );

OCPexport mpc(ocp);
mpc.set( HESSIAN_APPROXIMATION,GAUSS_NEWTON );
mpc.set( DISCRETIZATION_TYPE,MULTIPLE_SHOOTING );
mpc.set( INTEGRATOR_TYPE,INT_RK78 );
mpc.set( QP_SOLVER,QP_QPOASES );

if (mpc.exportCode( "./export" ) != SUCCESSFUL_RETURN)
exit( EXIT_FAILURE );
return EXIT_SUCCESS;
}
```

B. Simulation Plots

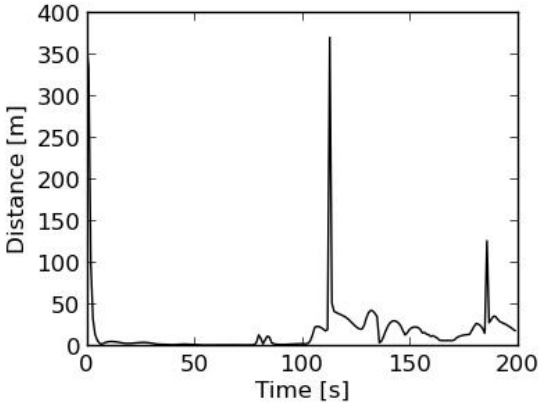
B.1. Too Small Offset on Heading in CM1



(a) East-north map.

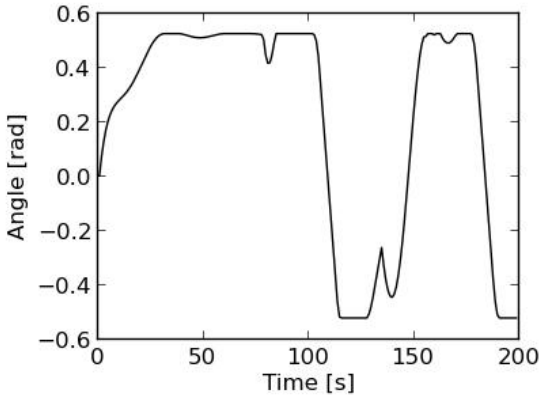


(b) CGP map.

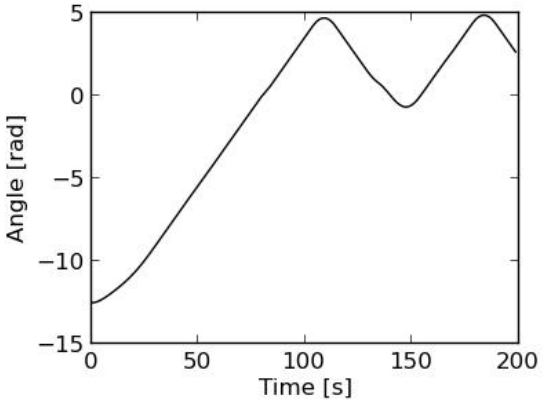


(c) Difference between CGP and object position.

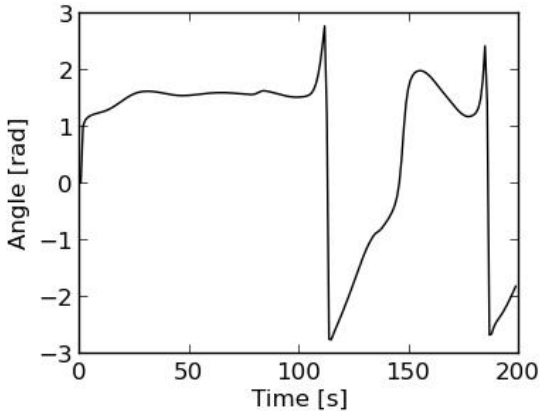
Figure B.1.: CM1 behaviour without disturbances and with a too small offset on heading.



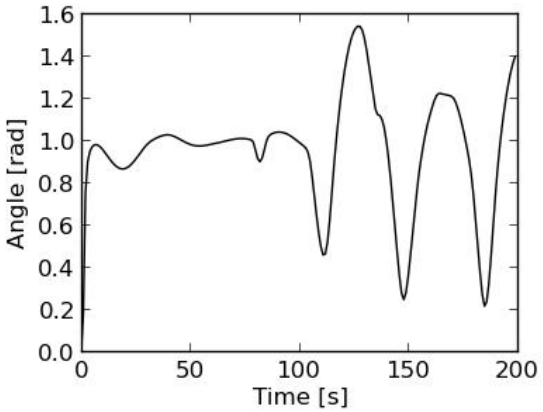
(a) Roll angle.



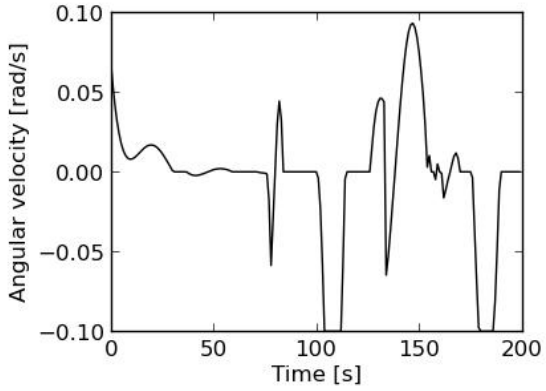
(b) Heading angle.



(c) Pan angle.



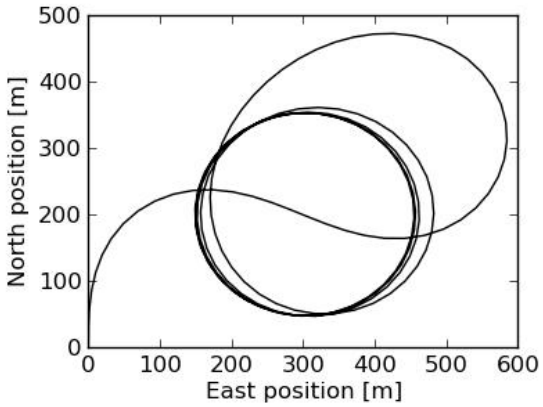
(d) Tilt Angle.



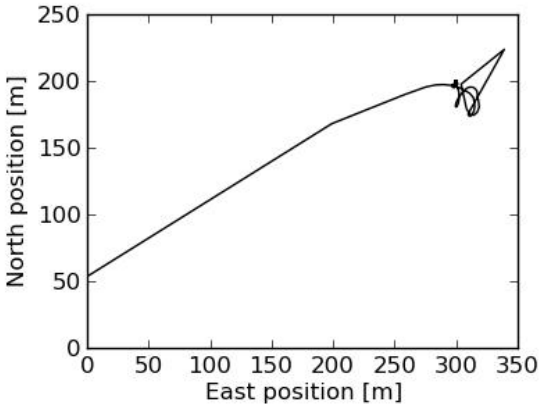
(e) Roll control input.

Figure B.2.: CM1 angles and control variable without disturbances and with a too small offset on heading.

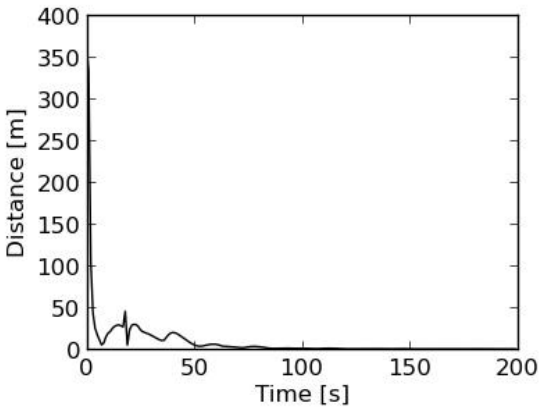
B.2. Large Positive Offset on Heading in CM1



(a) East-north map.

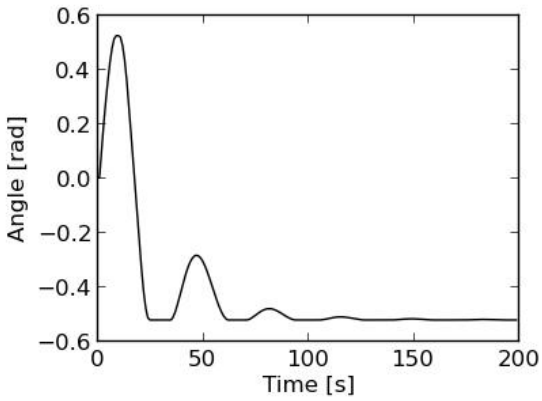


(b) CGP map.

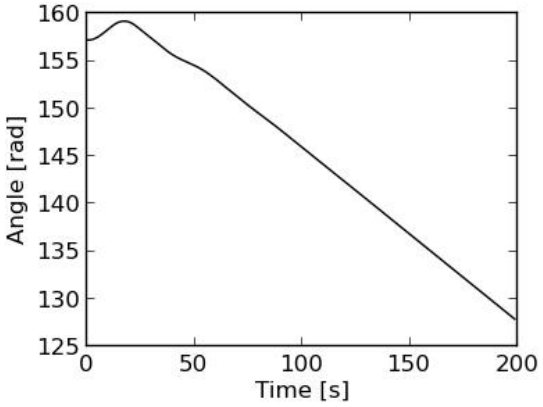


(c) Difference between CGP and object position

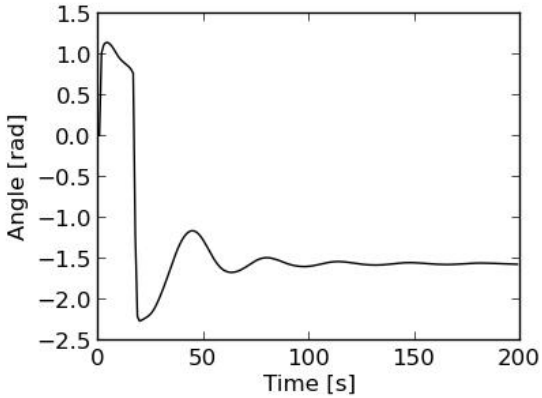
Figure B.3.: CM1 behaviour without disturbances and a large positive offset on heading.



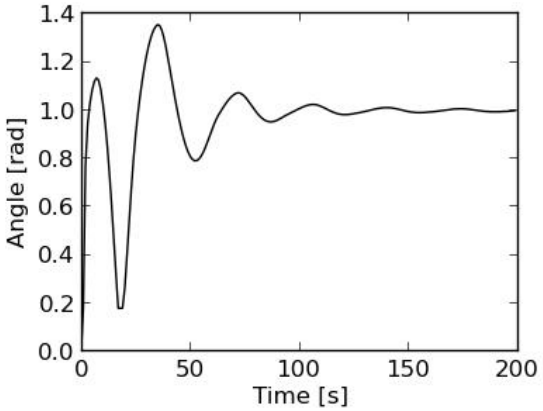
(a) Roll angle.



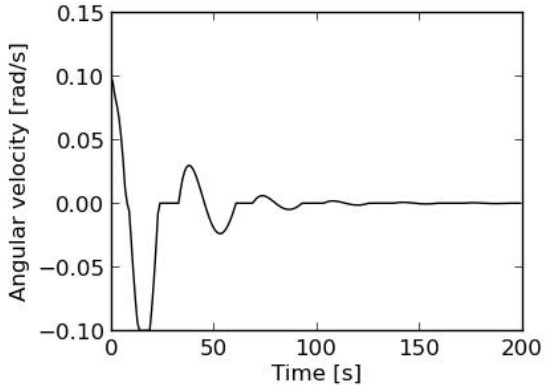
(b) Heading angle.



(c) Pan angle.



(d) Tilt Angle.

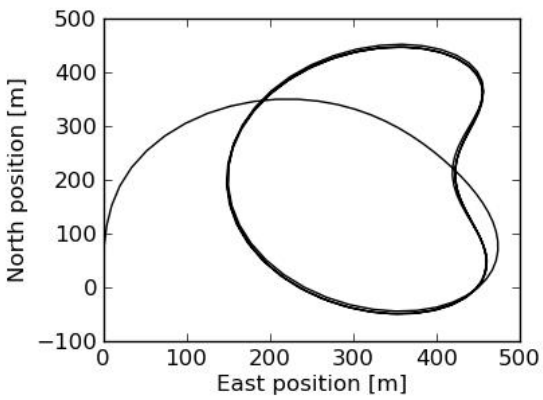


(e) Roll control input.

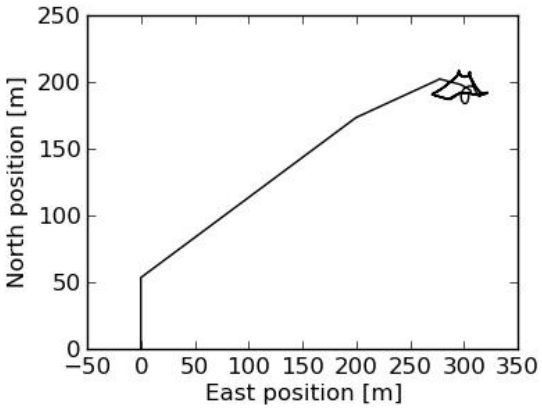
Figure B.4.: CM1 angles and control variable without disturbances and a large positive offset on heading.

B.3. Low Penalty on Roll Rate

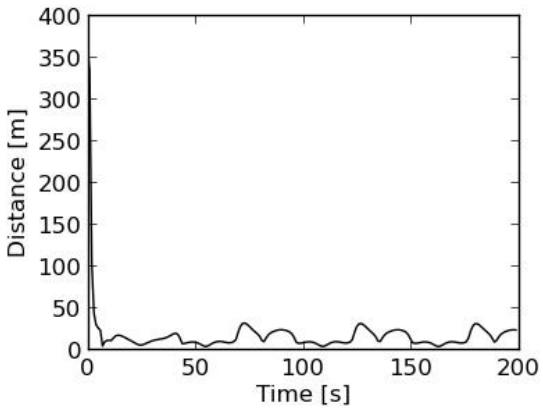
B.3.1. CM1



(a) East-north map.

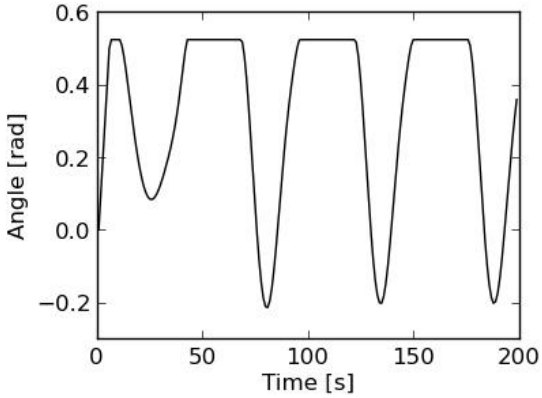


(b) CGP map.

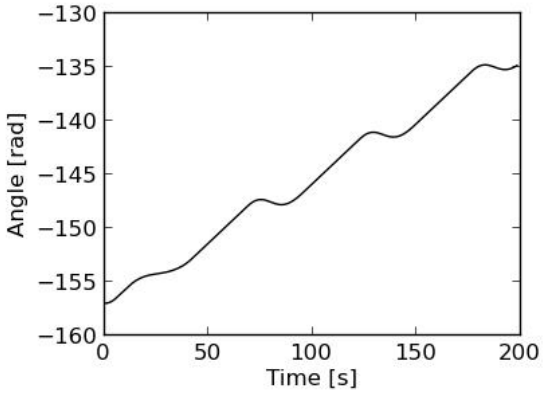


(c) Difference between CGP and object position.

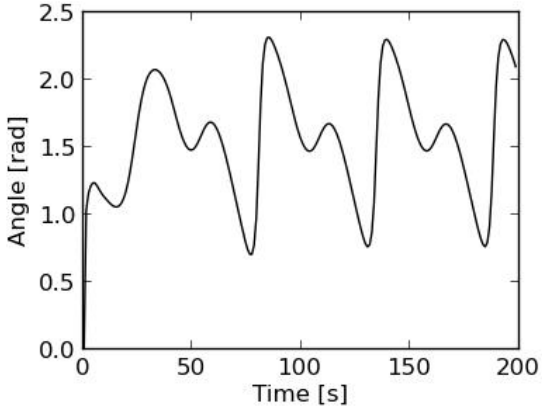
Figure B.5.: CM1 behaviour with wind disturbance and with a too small penalty on roll rate.



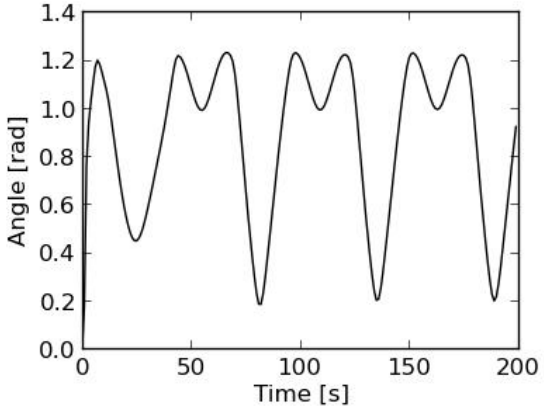
(a) Roll angle.



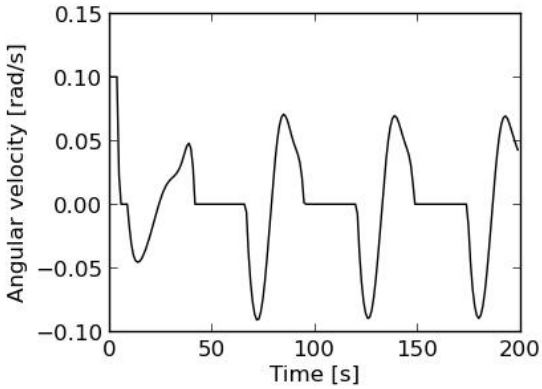
(b) Heading angle.



(c) Pan angle.



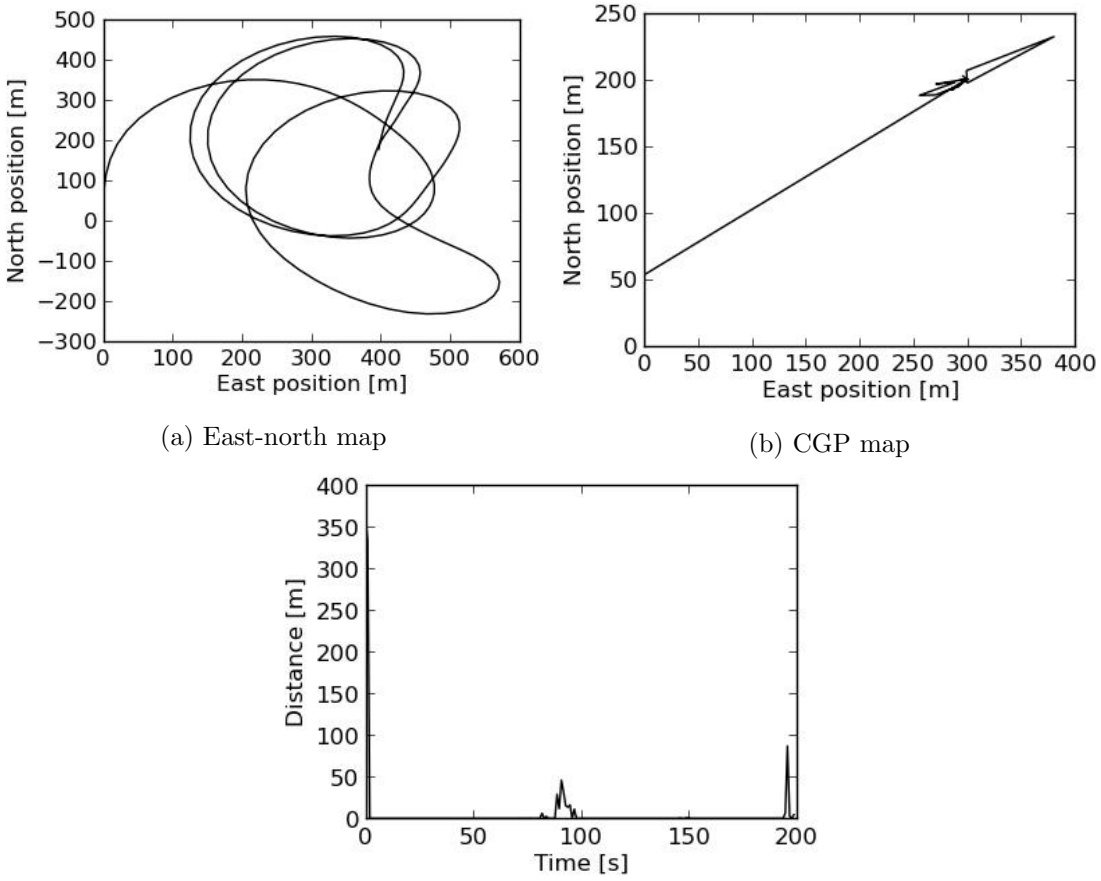
(d) Tilt angle.



(e) Roll control input.

Figure B.6.: CM1 angles and control variable with wind disturbance and with a too small penalty on roll rate.

B.3.2. CM3

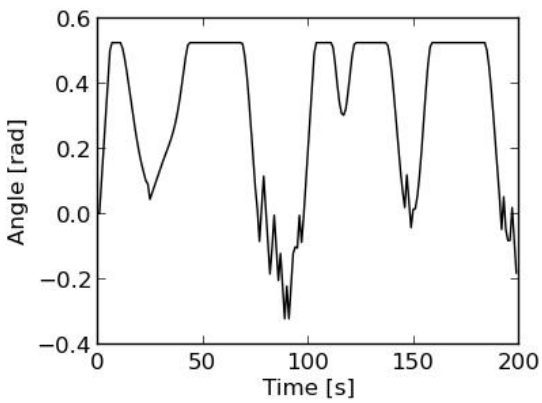


(a) East-north map

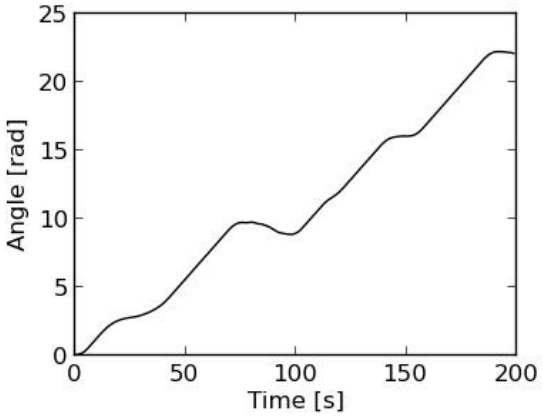
(b) CGP map

(c) Difference between CGP and object position.

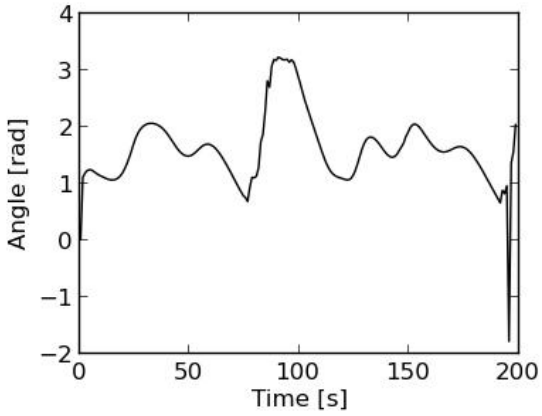
Figure B.7.: CM3 behaviour with wind disturbance and with a too small penalty on roll rate.



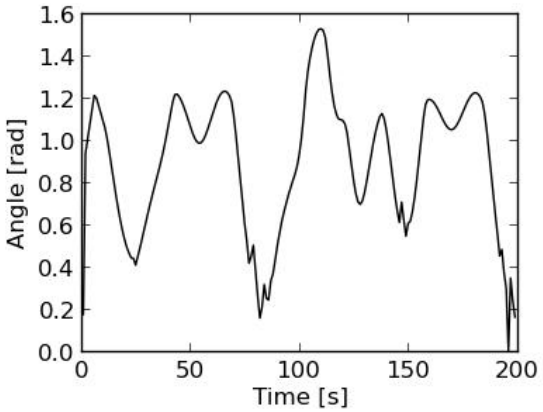
(a) Roll angle.



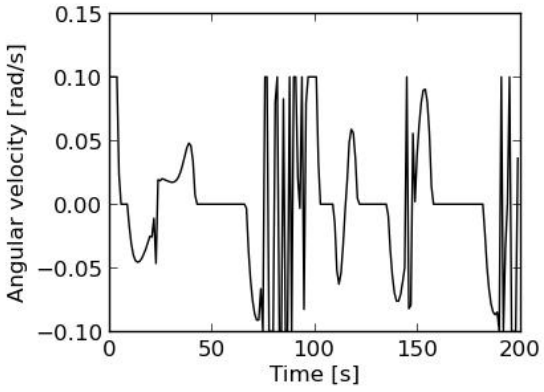
(b) Heading angle.



(c) Pan angle.



(d) Tilt Angle.



(e) Roll control input.

Figure B.8.: CM3 angles and control variable with wind disturbance and with a too small penalty on roll rate.

B.4. Time Usage on Laptop PC of CM3

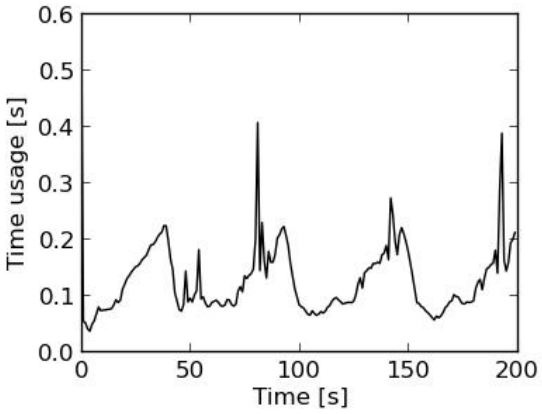


Figure B.9.: CM3 controller time usage on laptop PC.

C. HIL Plots

In this chapter, the results of the second HIL test taking pitch into account will be shown.

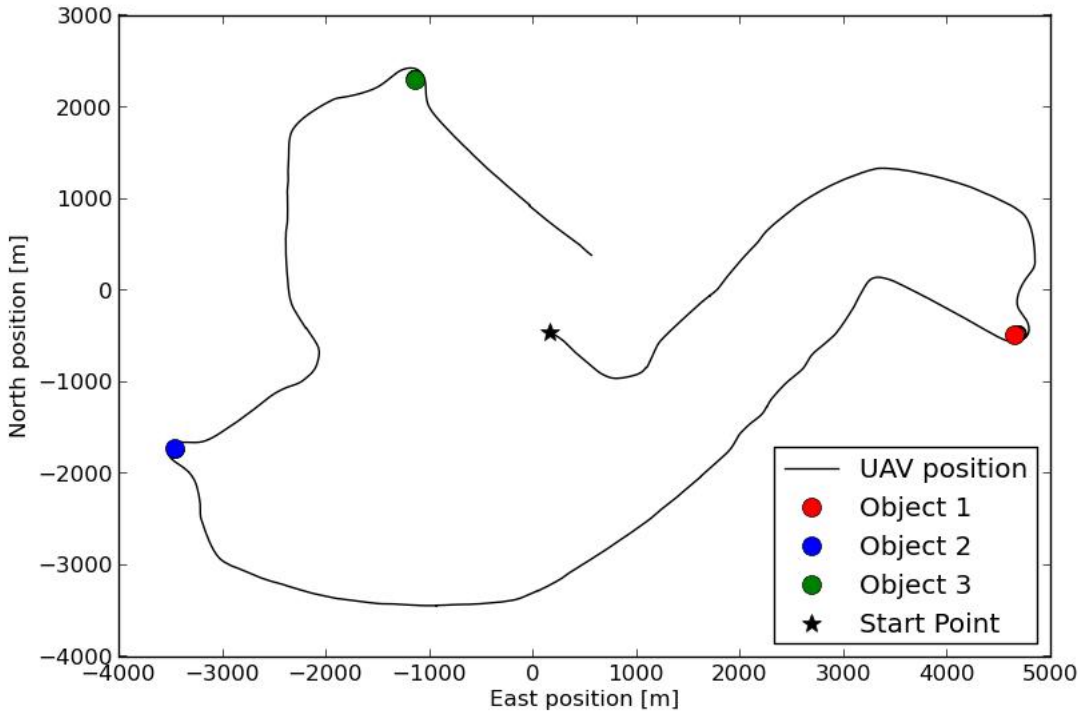
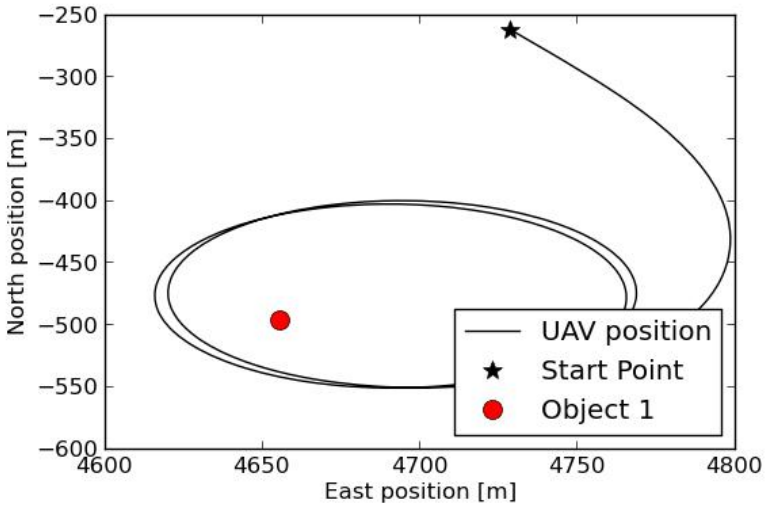
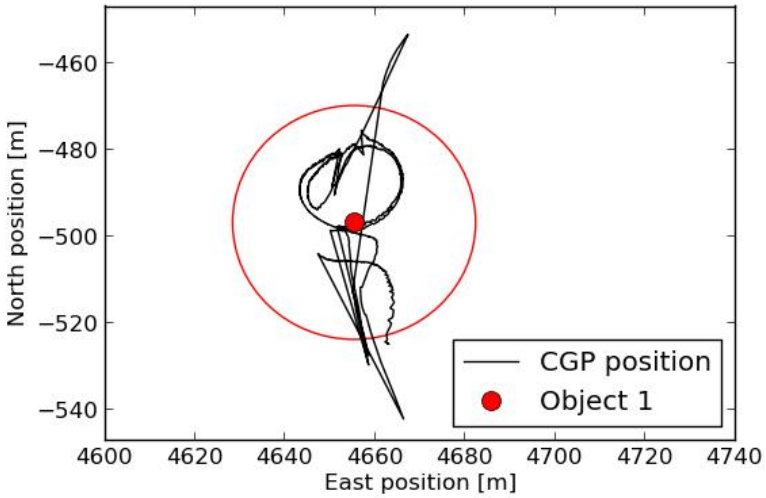


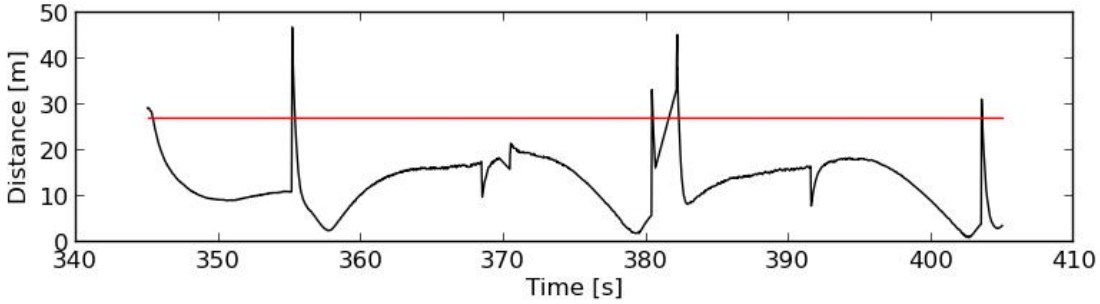
Figure C.1.: Flight path.



(a) Flight path near object 1 shown in east-north map.

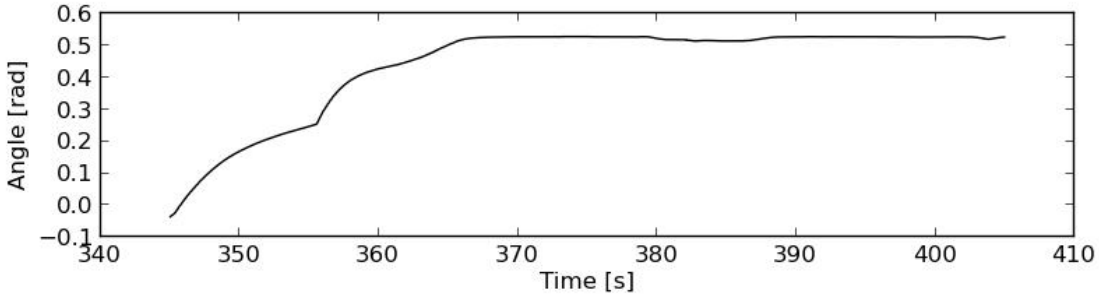


(b) CGP shown in east-north map when tracking object 1.

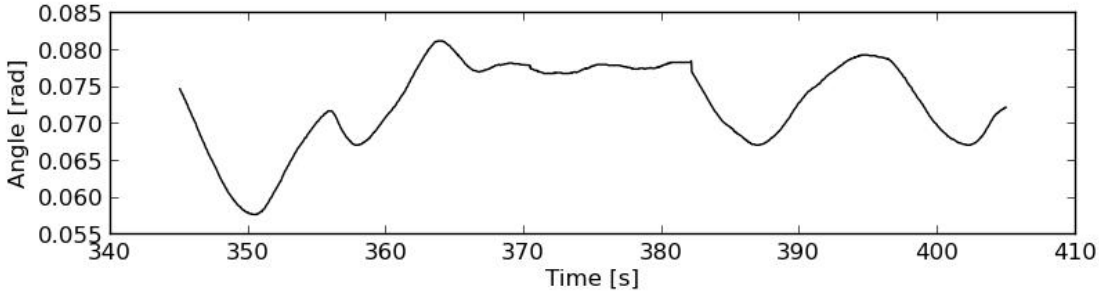


(c) Distance between CGP and object 1.

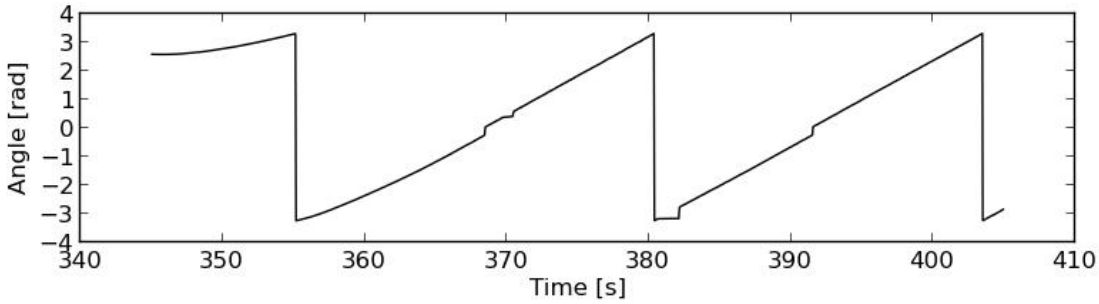
Figure C.2.: Behaviour near object 1.



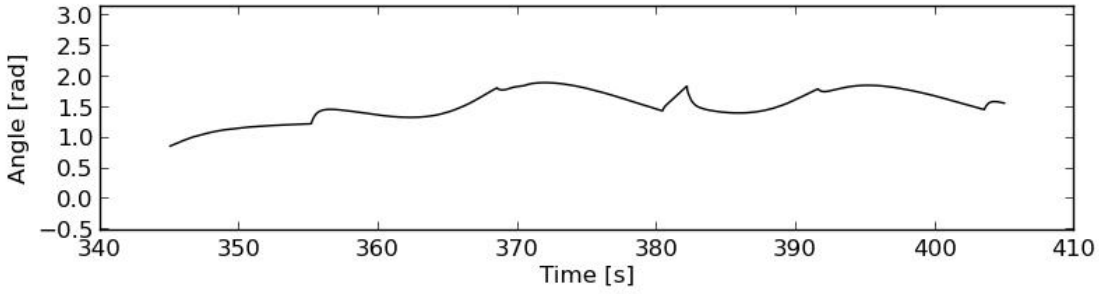
(a) Roll angle.



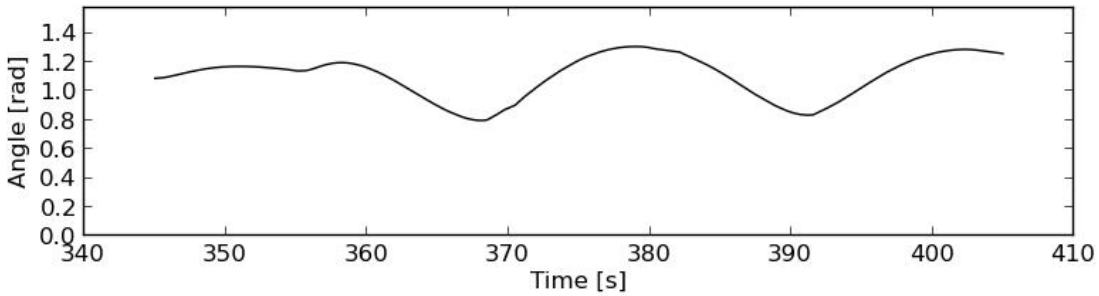
(b) Pitch angle.



(c) Heading angle.

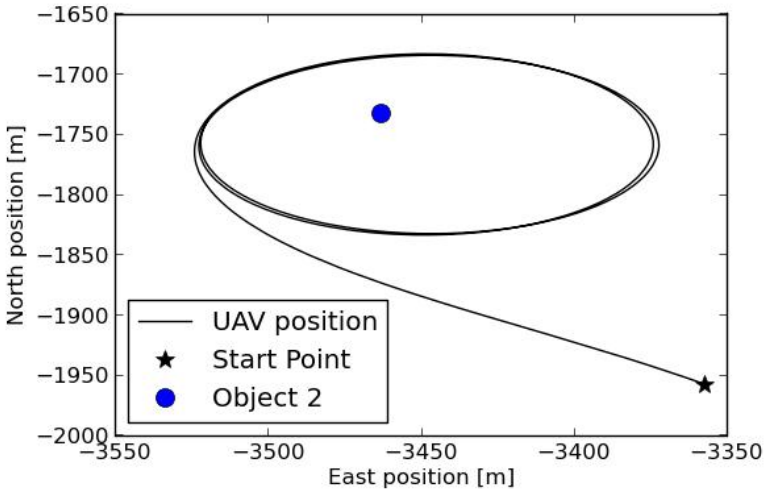


(d) Pan angle.

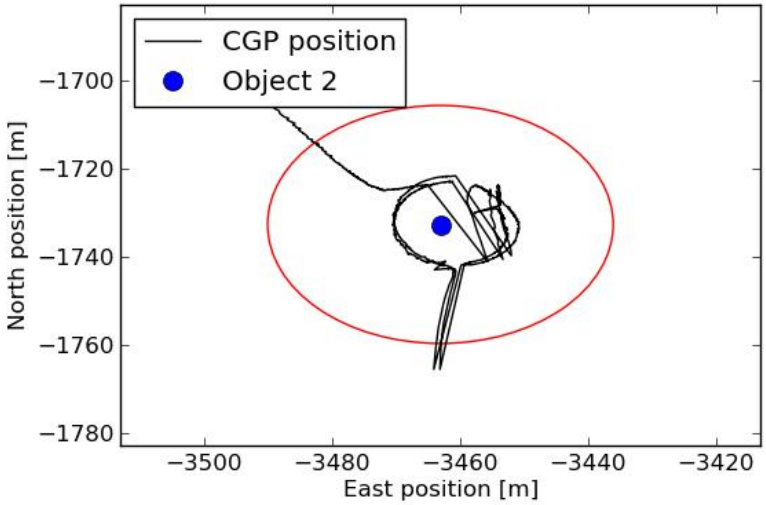


(e) Tilt Angle.

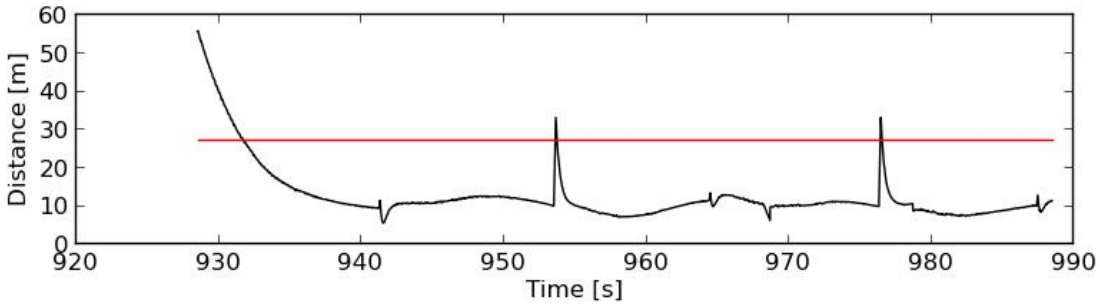
Figure C.3.: UAV and PTG angles near object 1.



(a) Flight path near object 2 shown in east-north map.

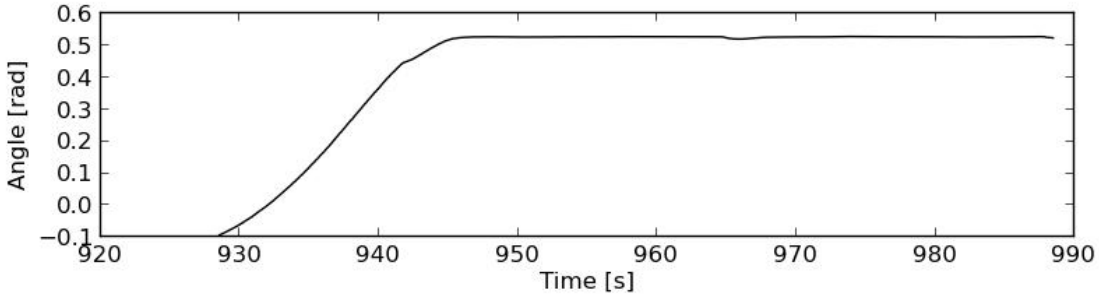


(b) CGP shown in east-north map when tracking object 2.

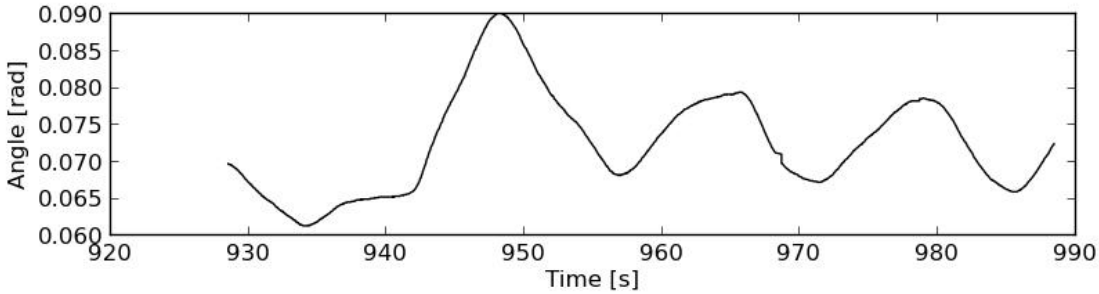


(c) Distance between CGP and object 2.

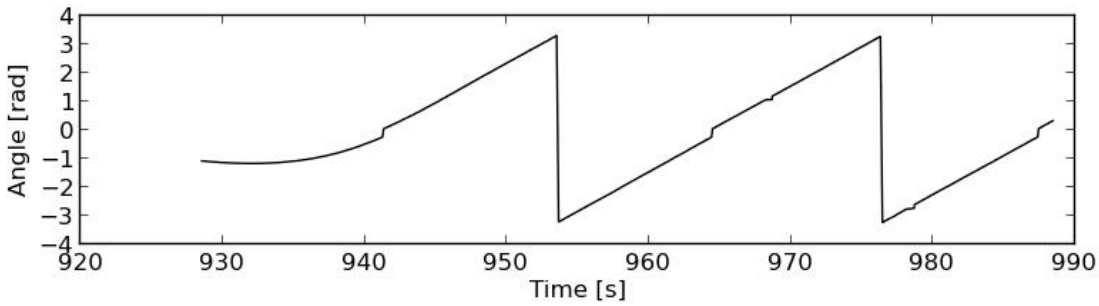
Figure C.4.: Behaviour near object 2.



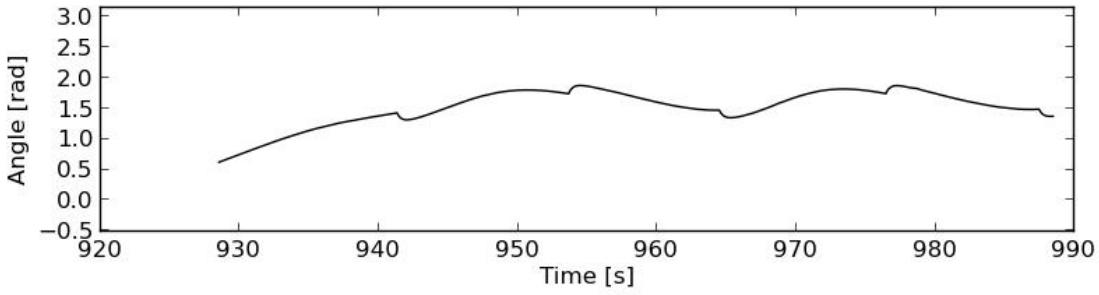
(a) Roll angle.



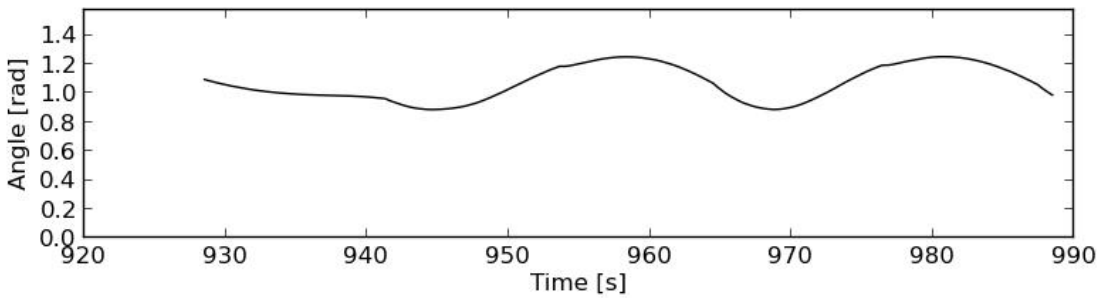
(b) Pitch angle.



(c) Heading angle.

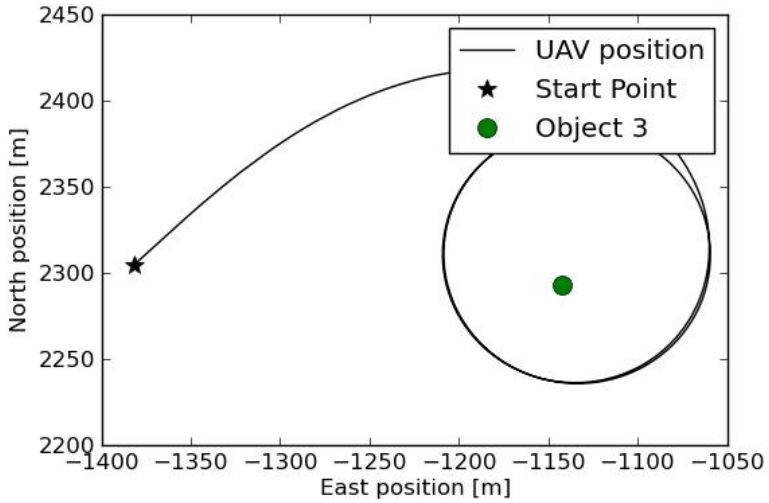


(d) Pan angle.

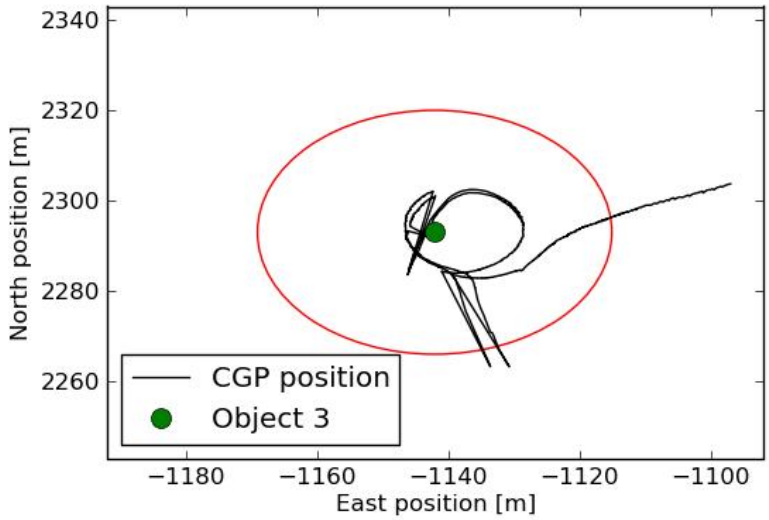


(e) Tilt Angle.

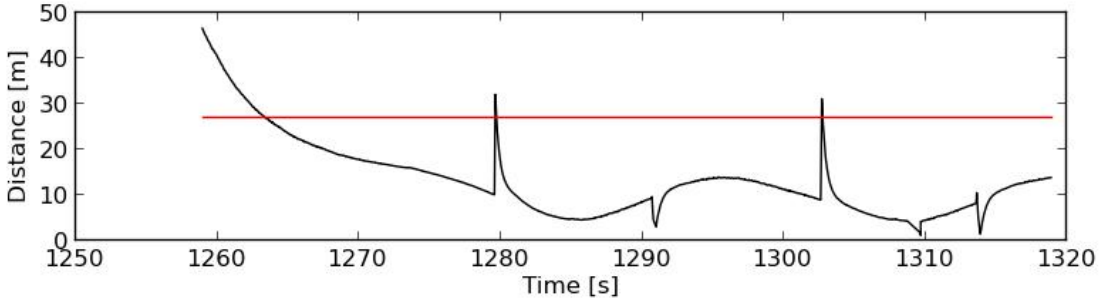
Figure C.5.: UAV and PTG angles near object 2.



(a) Flight path near object 3 shown in east-north map.

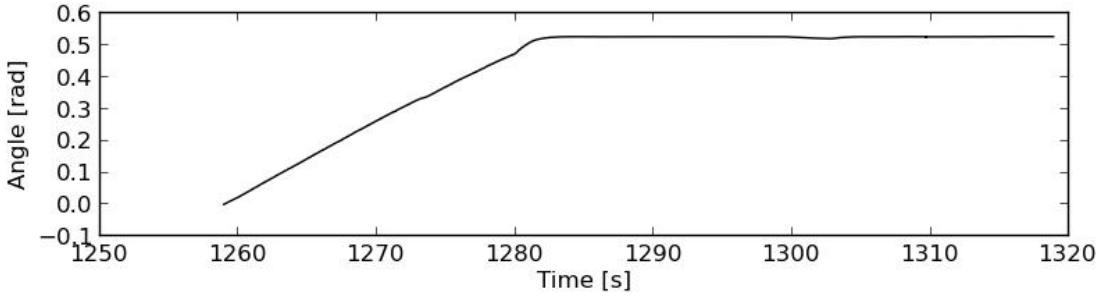


(b) CGP shown in east-north map when tracking object 3.

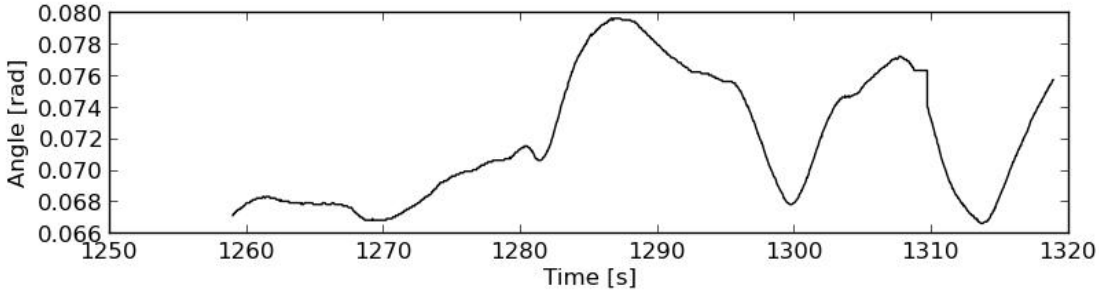


(c) Distance between CGP and object 3.

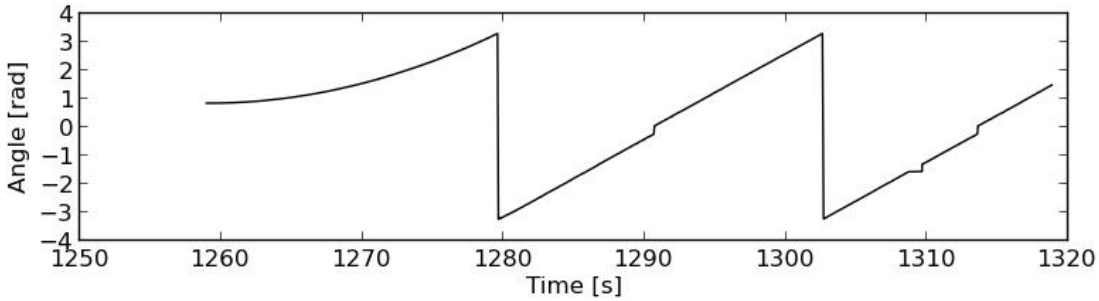
Figure C.6.: Behaviour near object 3.



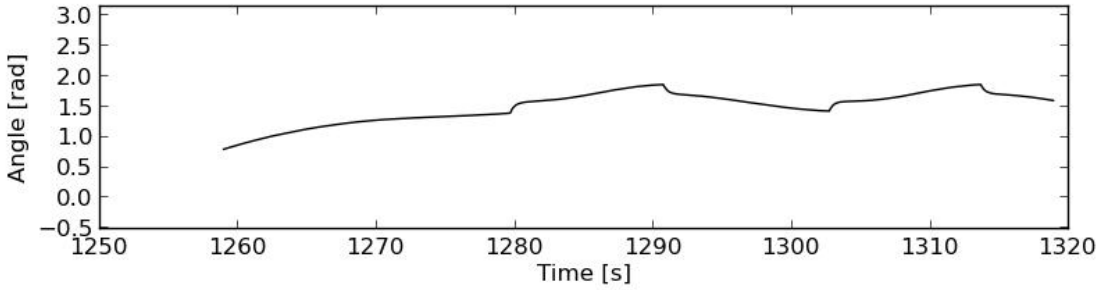
(a) Roll angle.



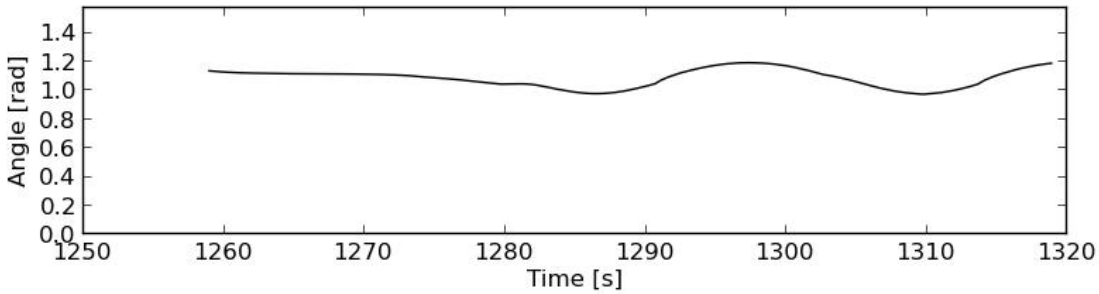
(b) Pitch angle.



(c) Heading angle.



(d) Pan angle.



(e) Tilt angle.

Figure C.7.: UAV and PTG angles near object 3.

D. Piccolo Command Center

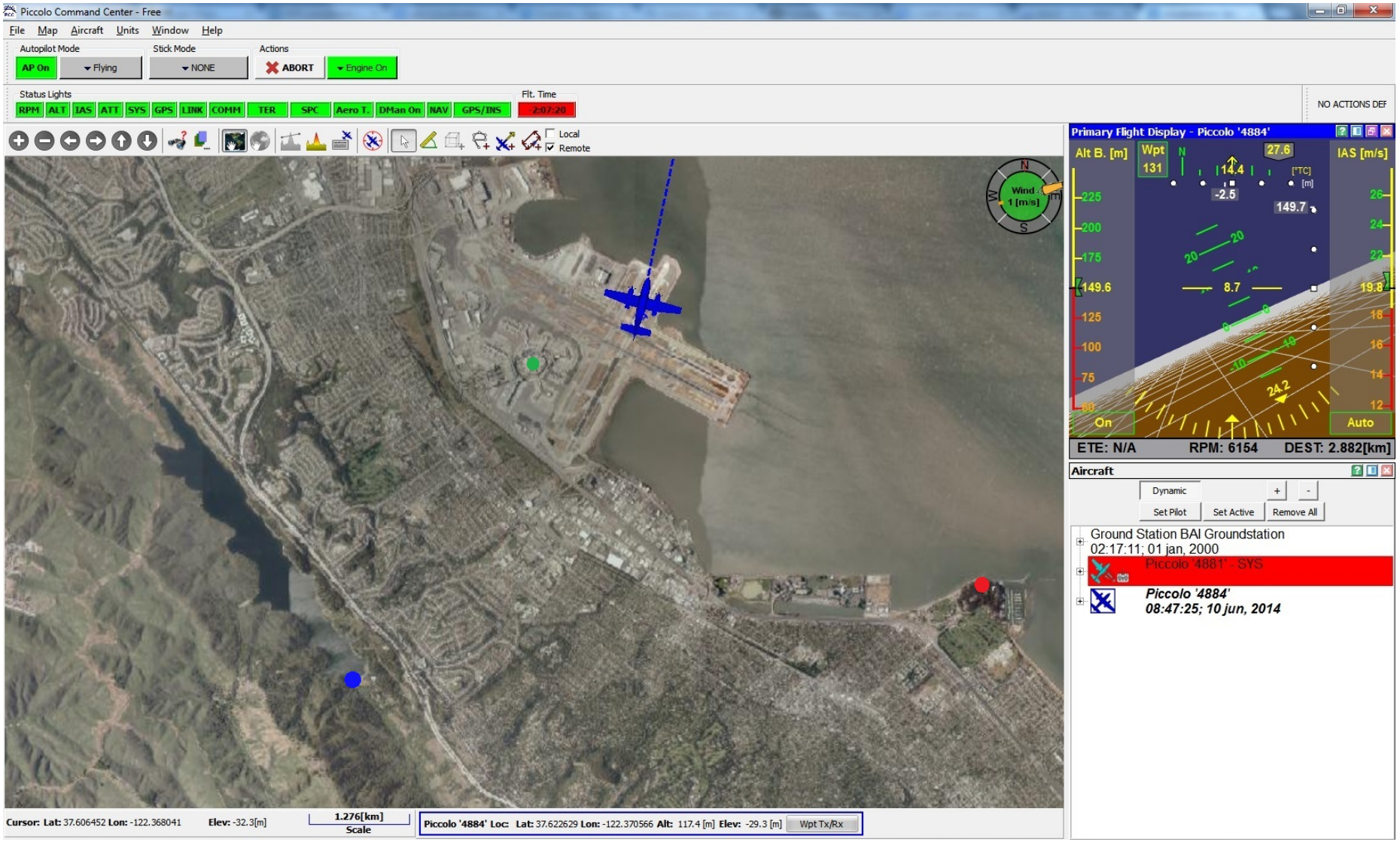


Figure D.1.: Image of the HIL simulation program in Piccolo Command Center. Object 1, 2, and 3, are indicated by the red, blue, and green dots, respectively.

E. Payload

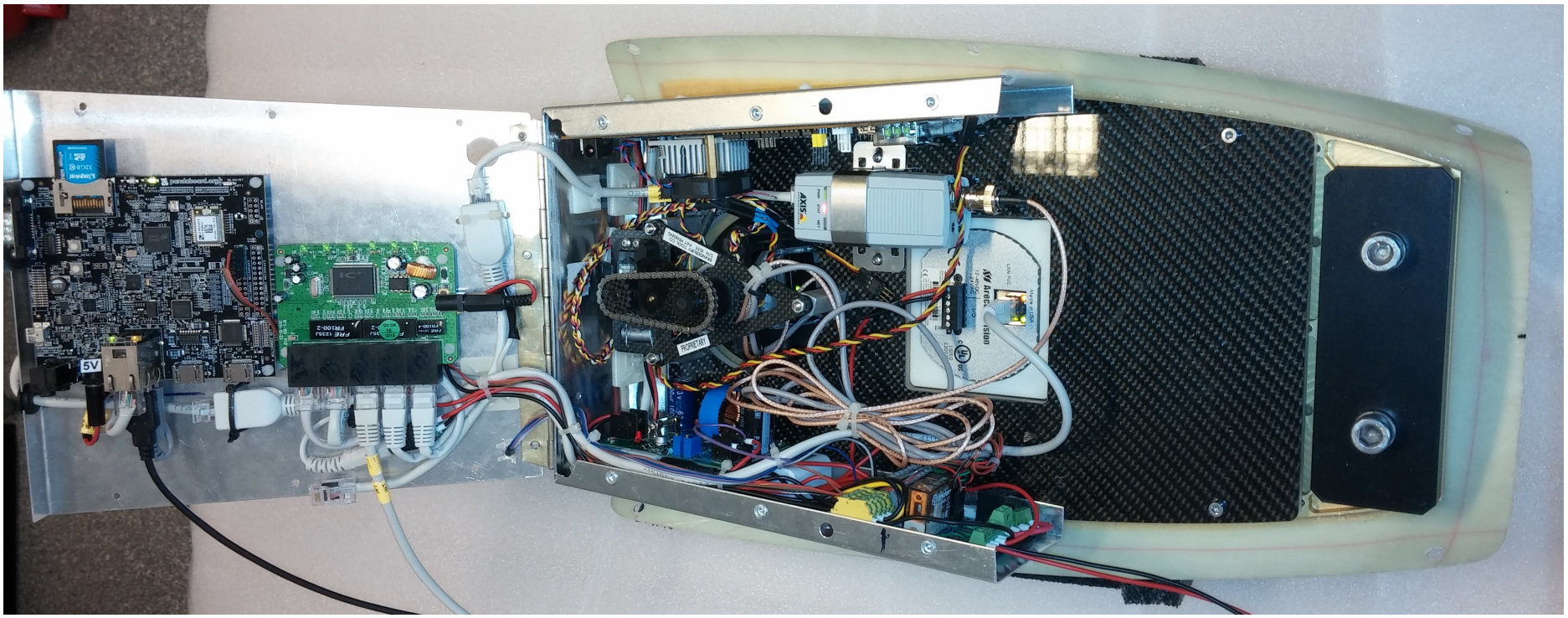


Figure E.1.: Image of payload on board the UAV. The payload is the metallic box. It stands on the underside of the UAV body, which is to be fastened to the UAV.

Bibliography

Axis Communications. Axis m7001 video encoder user's manual rev 2.0, July 2011. [23]

John Bellingham, Arthur Richards, and Jonathan P. How. Receding horizon control of autonomous aerial vehicles. In *Proceedings of the American Control Conference*, May 2002. [3]

Cloud Cap. Cloud cap technology piccolo sl, Jan 2014. [25]

FLIR. Tau 2 product specification, May 2014. [22]

Thor I. Fossen. *Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011. [10, 11, 33, 36, 81]

Thor I. Fossen. Mathematical models for control of aircraft and satellites. Technical report, NTNU, April 2013. [37]

Thor I. Fossen, June 2014. URL `http://www.itk.ntnu.no/fag/gnc/Wiley/Ch2.pdf`. [VII, 11]

S. Gratton, A.S. Lawless, and N.K. Nichols. Approximate gauss-newton methods for nonlinear least squares problems. Technical report, The University of Reading, April 2004. [16]

B. Houska, H.J. Ferreau, M. Vukov, and R. Quirynen. ACADO Toolkit User's Manual. http://www.acadotoolkit.org, 2009–2013. [13, 15]

MJ Mahoney, March 2014. URL `http://mtp.mjmahoney.net/www/notes/pointing/pointing.html`. [VII, 12]

Micro UAV. Btc-88 - micro pan / tilt unit, 2011. [23, 24]

pandaboard.org, 2014. URL `http://www.pandaboard.org/`. [22]

Rolf Rysdyk. Uav path following for target observation in wind. Technical report, 2006. [4]

Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modelling and Control*. Wiley, 2006. [7, 9]

Meirui Sun, Rongming Zhu, and Xueguang Yang. Uav path generation, path following and gimbal control, April 2008. [4]

Panagiotis Theodorakopoulos and Simon Lacroix. A strategy for tracking a ground target with uav. Technical report, University of Tolouse, 2006. [4]

UAV Factory. 3w 28i / 28cs engine with 80w generator system. Technical report, May 2014a. V 2.0. [20]

UAV Factory. Penguin b uav platform datasheet v 2.0, May 2014b. [19]

Ubiquiti Networks. Rocket m5 technical specs / datasheet, May 2014. [25]

U.S. Department of Defence. Dictionary of military and associated terms, April 2005. [1]

Bjørnar Vik. *Integrated Satellite and Inertial Navigation Systems*. Department of Engineering Cybernetics, NTNU, 2012. [33]

what-when-how.com, June 2014. URL `http://what-when-how.com/advanced-methods-in-computer-graphics/kinematics-advanced-methods-in-computer-graphics-part-1/`. [VII, 7]

Kwangjin Yang and Salah Sukkarieh. Adaptive nonlinear model predictive path tracking control for a fixed-wing unmanned aerial vehicle. In *AIAA Guidance, Navigation, and Control Conference*, August 2009. [3]