Accurate Drop of a GPS Beacon Using the X8 Fixed-Wing UAV

Vegard Grindheim

Master of Science in Cybernetics and Robotics
Submission date: June 2015
Supervisor: Thor Inge Fossen, 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:	Vegard Grindheim
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Nøyaktig slip av GPS transmitter fra et X8 ubemanned fly
Thesis title (English):	Accurate Drop of a GPS Beacon Using the X8 Fixed-Wing UAV

Thesis Description: The purpose of the thesis is to develop a system for high-precision deployment of a GPS based position transmitting beacon from a small fixed-wing UAV in an Arctic environment. The system will be able to hit sea ice consistently and have wind estimation and compensation to achieve the required accuracy.

The following items should be considered:

1. Implement and test a wind estimator.
2. Given a desired target position for the beacon to land at, find all feasible aerial release points (FARP) and their corresponding velocities.
3. Design a path planner based on the Adapted Dubins Path algorithm and test the system by performing software-in-the-loop (SIL) and field tests.
4. Design and implement a path planner using optimization techniques to choose a computed aerial release point (CARP) within the FARP, and the path to get there. Verify the controller and optimization results with simulations and HIL testing.
5. Implement a line-of-sight (LOS) controller that guides the UAV along a path to the chosen CARP.
6. Research different combinations of path planners, guidance controllers and wind estimators to reach the CARP.
7. Verify the controller, path planner and optimization results with SIL testing.
8. Test the UAV system in the field.
9. Compare the results of the approach methods and conclude your results.

Start date:	2015-01-05
Due date:	2015-06-14

Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Professor Thor I. Fossen, Dept. of Eng. Cybernetics, NTNU
Co-Supervisor:	Siri Mathisen, Dept. of Eng. Cybernetics, NTNU

Preface

Researchers at the Centre for Autonomous Marine Operations and Systems (AMOS) work on making unmanned aerial vehicles (UAVs) conduct inaccessible, dull, costly and dangerous missions. This report was written in cooperation with the UAV-lab research group at AMOS in the final semester of my two-year master's course in Engineering Cybernetics. The main subject of this thesis has always been a passion of mine, ever since I was a teenager.

At the age of 14, I was finally able to acquire a real flying model aeroplane after accumulating several years worth of savings. It was a small model of the Piper J-3 Cub. That plane took me about half a year to build, and took up most of the table space in the TV room for the entire time. The only downside about the plane was, according to that 14 year old version of me, that there was no way to drop model bombs with it.

At this day and time though, I hope that my work will never be used for any sort of military action, but that it may be used for civil applications.

That aeroplane is now far gone, crashed beyond the scope of what any amount of duct tape might repair. If you think that it lived a long and prosperous life, you are wrong. Teenagers being teenagers led to acrobatic flying reserved for entirely different model planes. I never once returned home with the plane in one piece. Four pieces was far more likely.

That first experience with an aeroplane has a lot in common with this thesis. There was a lot of work, both include flying, crashes did occur, and both were, at the time, the biggest project I had ever partaken in. Now I only hope the knowledge of this thesis might last longer than my aeroplane did.

This report assumes that the reader has a general background in control engineering.

Vegard Grindheim

Trondheim, June 12, 2015

Acknowledgments

I would like to thank my supervisors, Thor Inge Fossen and Siri Holthe Mathisen, for guidance and support throughout my work on this thesis. It would not have been the same without them. I would also like to thank Kristian Klausen and João Fortuna for great technical support, especially their help with programming in DUNE and getting the payload computer up and running. Last but not least, I wish to thank Torkel Hansen and Lars Semb, who have guided me with practical aspects related to the UAV and release mechanism, and of course, piloting the aircraft.

V.G.

Summary

Improved possibilities for transport, as a consequence of diminishing ice mass in the Arctic, leads to more activity in these areas. Some of the greatest dangers of heightened presence in the Arctic is associated with sea ice.

To decrease the dangers of sea ice in Arctic areas, fixed-wing UAVs can be used to find and tag dangerous ice with GPS based position transmitting beacons. This would allow traffic and offshore installations to operate more safely by knowing about dangers beforehand.

This thesis will focus on the specific task of making a UAV system capable of hitting sea ice with a GPS beacon by an unguided air drop. This task has several parts, including calculating the aerial release point and guiding the UAV to this point with high accuracy. The combined drop accuracy and precision required from this system is to be able to hit sea ice with a radius of 10 meters 95% of the time. 10 m is approximately the radius of a small floe.

Several approach methods were applied with different controllers and path planners to accurately guide the UAV to the aerial release point. The best combination of these used optimization techniques combined with a line-of-sight path-following controller. This combination acquired a theoretical accuracy of 1.63 m and a precision of 7.14 m from simulated results. This was combined to give a 95% chance of hitting within 8.77 m of the sea ice centre, which is better than the required accuracy set in this thesis.

This result was unfortunately not confirmed by flight tests, as unforeseen circumstances prevented flight time when the system was completed. However, early flight tests validates the correctness of the simulation results to some degree.

Sammendrag

Forbedrede transportmuligheter som følge av lavere ismasse i Arktis fører til økt virksomhet i disse områdene. Noen av de største farene ved aktivitet i Arktiske omgivelser er knyttet til sjøis.

For å minske farene ved sjøis i arktiske områder kan fastvinge-UAV-er brukes til å finne og merke farlig sjøis med GPS-baserte posisjonstransmittere. Dette vil føre til økt sikkerhet for trafikk og offshoreinstallasjoner ved å informere om fare før den inntreffer.

Denne masteroppgaven fokuserer på den spesifikke oppgaven å lage et UAV-system som kan treffe sjøis med posisjonstransmitteren ved bruk av et fritt fall. Oppgaven består av flere deler, inkludert å beregne slippunktet i luften og å guide UAV-en til dette punktet med høy nøyaktighet. Den forventede kombinerte nøyaktigheten og presisjonen til dette systemet er å kunne treffe sjøis med en radius på ti meter med minst 95% sjanse. Ti meter er omtrentlig radiusen til et lite isflak.

Flere tilnærmingsmetoder ble brukt med forskjellige regulatorer og ruteplanleggere for å styre UAV-en til slippunktet. Den beste kombinasjonen brukte optimaliseringsteknikker sammen med en siktlinje-rutefølgende regulator (line-of-sight path-following controller). Denne kombinasjonen oppnådde en teoretisk nøyaktighet på 1.63 m og en presisjon på 7.14 m fra simulerte resultater. Disse ble kombinert til å gi 95% sjanse for å treffe innen 8.77 m av isens midtpunkt, hvilket er et bedre resultat enn kravspesifikasjonen.

Dette resultatet kunne dessverre ikke bekreftes gjennom testflyvninger, ettersom uforutsette forhold forhindret flyvninger etter at systemet var ferdigstilt. Tidlige testflyvninger validerer derimot simuleringsresultatene til en viss grad.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The thesis is presented in this chapter, its background and motivation explained and the problem formulated and described in separate sections. Papers and books used in the literature study is discussed. Finally, the structure of the rest of the thesis is described.

1.1 Background

The possibilities for unmanned aerial vehicles (UAVs) are endless. By using these lightweight, low cost vehicles it is possible to both explore new fields of operation and support existing industries. The main advantage about UAVs is that they are able to perform missions and tasks that may otherwise be dull, dangerous or expensive to conduct. As industry expands northward and more activities takes place in the Arctic, UAVs can be used for support.

Operational support for the maritime industry, where ice properties or search and rescue missions may be of great interest (Westall et al., 2007), is particularly viable. Other examples are emergency preparedness and response due to oil spills (Puestow et al., 2013), volcanic ash concentration or other industrial accidents. The petroleum industry could also have a great benefit of using UAVs for mammal detection, sea-ice surveillance and ice-berg detection (Eik, 2010).

As the industry expands and more activities takes place in the Arctic, localization of drifting glacial ice will be of great interest (Eik, 2010). There are many ways of achieving the needed overview, but these methods are often costly or currently unavailable. One low-cost way to do this is to drop GPS based position transmitting beacons (henceforth referred to as beacons) onto the drifting ice in order to keep track of its location (Shestov and Marchenko, 2014). This is possible to do by using UAVs. This strategy will minimize time and cost of deployment of the beacons compared to other methods (McGill et al., 2011). As described in both the master theses Mathisen (2014) and Fuglaas (2014), and in McGill et al. (2011) there are no current off-the-shelf product for dropping payloads from UAVs. However, the need for such methods are increasing as e.g better practise in ice advising is needed with the increasing amount of Arctic operations (Tiffin et al., 2014).

Automating such a process is quite advanced and it consists of multiple objectives:

- Searching for glacial ice
- Take off and landing
- Dropping the beacon

Each of these objectives has their own challenges. This thesis will focus on the objective of dropping a beacon by using a fixed-wing UAV.

In order to drop the beacon as accurately as possible, a precise knowledge of the UAV's position, attitude and forces acting on the UAV is needed. The forces will also affect the beacon while falling. Using an on-board UAV sensor suite and GPS technology, the environmental disturbances can be estimated (Johansen et al., 2015; Langelaan et al., 2011).

1.2 Problem Formulation

The purpose of the thesis is to develop a system for high-precision deployment of a GPS based position transmitting beacon from a small fixed-wing UAV in an Arctic environment. The system shall be able to hit sea ice consistently and have wind estimation and compensation to achieve the required accuracy.

1.3 Literature Survey

All the appropriate material that was discovered during the literature survey will be summarized concisely in the following sections.

1.3.1 Air-Drop Systems

Air-drop systems is a widely explored subject. Its use is mainly military, where a broad variation of air-drop methods have been considered (Goldfein, 2013, ch. IV p. 86-91). These methods vary from personal air-drop systems to free-fall methods. In order not to supersede the boundaries of this thesis, the focus of this report will be on the methods that are meaningful for the task at hand.

Guided Air Drop

There are many different methods of guided air-drop delivery. The most normal techniques consist of using various kinds of parachutes, with and without control systems, to safely deliver equipment to the ground. Deployment by parachute is described by literature in Henry et al. (2010) and Wuest and Benney (2005). One can also utilize cable guided system (Mathisen, 2014), such as ground anchored cables or wired coils.

Free Fall Air Drop

This gravity-based parachute-less method uses information about initial conditions of the UAV and beacon to find the calculated-aerial-release point (CARP) (Wuest and Benney, 2005). In addition to the initial conditions, wind compensation is required in order to compensate for the environmental effects. The wind compensation in Wuest and Benney (2005) assume mean effective wind forces, which translates to inaccuracy. This was further investigated in Fuglaas (2014). MATLAB simulations and software-in-the-loop (SIL) tests were utilized in order to produce estimates of the wind uncertainty. The conclusion of these simulations show that the higher the release height, the larger the impact distribution. Simulations also showed that an imperfect release configuration, leading to varying release-time, also produced an uncertainty.

The study of Fortier (2004) also concludes that dominant errors are the release configuration and altitude and wind knowledge. Fortier (2004) considers wind-estimation methods and wind characteristics, based on general wind statistics and wind forecasting statistics. The wind forecasting statistics are retrieved from weather forecast and weather balloons equipped with GPS. The weather balloons are released in close proximity of the target at the time of a free fall drop. Data from the weather balloon is then used to correct the weather forecasts.

1.3.2 Wind Estimation

As described above, one of the main errors for air-drop methods is wind knowledge. A knowledge of wind velocity may be valuable information in order to calculate the CARP accurately (Fuglaas, 2014). Bencatel and Girard (2011) explains a method for estimation of the surface shear wind using a UAV equipped with a particle filter. This particle filter produces estimations of the wind speed and direction affecting the UAV.

Both Johansen et al. (2015), Brezoescu (2014) and Langelaan et al. (2011) explain methods for on-line wind estimation using a small fixed-wing UAV. This is obtained from either attitude dynamics or vehicle velocity by using on-board sensors.

1.3.3 Ballistic Paths

The ballistic path for a free-falling object can be calculated by using Newton's second law while taking drag force and gravity into account (Parker, 1977; Sherwood, 1967). Both Parker (1977) and Sherwood (1967) emphasizes the role the drag force has on a moving object.

1.4 Objectives

The main objectives of this thesis are:

1. Implement and test a wind estimator.
2. Given a desired target position for the beacon to land at, find all feasible aerial release points (FARP) and their corresponding velocities.
3. Design a path planner based on the Adapted Dubins Path algorithm (Fuglaas, 2014) and test the system by performing software-in-the-loop (SIL) and field tests.
4. Design and implement a path planner using optimization techniques to choose a computed aerial release point (CARP) within the FARP, and the path to get there.
5. Implement a line-of-sight (LOS) controller that guides the UAV along a path to the chosen CARP.
6. Research different combinations of path planners, guidance controllers and wind estimators to reach the CARP.
7. Verify the controller, path planner and optimization results with SIL testing.
8. Test the UAV system in the field.
9. Compare the results of the approach methods and conclude your results.

1.5 Limitations

As the UAV is required to be able to hit sea ice consistently, the required drop accuracy and drop precision in this thesis is therefore set to ten meters around the target position. This matches the size of a small floe (Bushuyev, 1970).

Considerations based on the literature survey and physical limitations of the UAV led to the conclusion that an unguided free-fall would be the air-drop method of choice. This decision was taken as the guided fall is too complex and does not guarantee better accuracy. Also, a parachute guided system could influence the back mounted motor propeller on the UAV.

The main bulk of the equipment that was used was chosen based on what was available at NTNU's UAV lab. Price was also taken into account to some extent. To simplify modifications, such as fitting the drop mechanism, a styrofoam airplane was chosen. Based on the above circumstances, propulsion was decided to be electric and weight became a critical aspect. Computational power was also sparse to save battery, weight and funds. The project was also supposed to use a standard UAV instrument suite.

The available fixed-wing UAV has a back mounted propeller, with which the beacon must not interfere. It was decided that the engine must be stopped during a drop to achieve this.

1.6 Contributions of This Thesis

With regards to the problem formulation (Section 1.2), objectives (Section 1.4) and limitations (Section 1.5) this thesis can be summarized as follows.

The focus of this thesis is to develop a means for high-precision, free-fall air-drops using a simple electric fixed-wing UAV. It will investigate possibilities for wind estimation, and avoid wind effects.

The specific contributions of this thesis are listed below

Approach Methods The thesis will explore several ways to approach the target using different path planners, flight controllers and wind estimators.

Path Planners Path planners will be designed, implemented and tested in SIL.

Wind Estimator from Johansen et al. (2015) Implementation and testing of the wind estimator will be performed.

FARP The feasible-air-release-points shall be defined, as well as an iterative solution to finding them.

Optimization on the FARP An optimization will be designed and implemented to find the best CARP in the FARP.

CARP Detection A crossover between standard perpendicular half-plane-crossing detection and spherical detection will be used to decide when the CARP is reached.

New Tasks Some tasks will be designed, implemented and tested. This will include, for example, a state machine meant to be running on the UAV during a GPS drop mission.

Testing and Concluding from the Results Accuracy and precision will be calculated through testing of all approach methods, wind estimators, controllers, CARP detection, state estimates and computational lags. This will be used to calculate the real world accuracy and precision to the system's ability to hit the target.

1.7 Structure of the Report

The first chapter of this thesis contains an introduction to its contents and summarizes its contributions. In Section 1.1, reasons for developing and using a UAV system for free fall drop was established. The specific objectives can be found in Section 1.4.

Theory used throughout the thesis was established in Chapter 2. All specific theory for Objective 1 to 6 is described here.

The development and choice of hardware and software are described in Chapter 3 and 4, while the final setup and installation is described in Section 3.4 and Section 4.5, respectively. This includes the Pixhawk, Dune, drop mechanism, GPS and communication devices. The implementation of the wind estimator in Objective 1 and the approach methods in Objective 6 are described in Section 4.4.2 and Section 4.4.3, respectively.

Testing is described in Chapter 5 and 6, and is divided in simulation and flight testing. The SIL testing in the simulation part fulfils the requirements set in Objective 1, 3 and 7, while the flight test fulfils Objective 3, but not 8, as no flight testing of the Optimal or OWSI path planner was performed.

The discussion is placed in Chapter 7, where the test methods and the complete system is debated. The comparison described in Objective 9 is found in Section 7.5.

Chapter 2

Theory

This chapter gives a detailed description of the theory and background material used in the rest of the thesis.

It is important to know when and where to release the beacon to accurately hit the target. The calculations required to find the CARP includes drag and gravity. This will be discussed in Section 2.1. To know how the UAV will behave, its kinematics are explained in Section 2.2. An explanation of the different coordinate systems is also included in this part. Some sources of error in payload drops are wind and position measurement errors. Wind estimation is therefore discussed in Section 2.3. To reach the CARP with accuracy, several approach strategies are tested in this thesis. An approach strategy is a specific combination of a path planner and a flight controller. To guide the UAV, flight controllers are used. The different controllers are described in Section 2.4. The path planner theory is found in section 2.5. As the need for accuracy is acute, some way of detecting the CARP accurately is needed. The detection used in this thesis is found in Section 2.6.

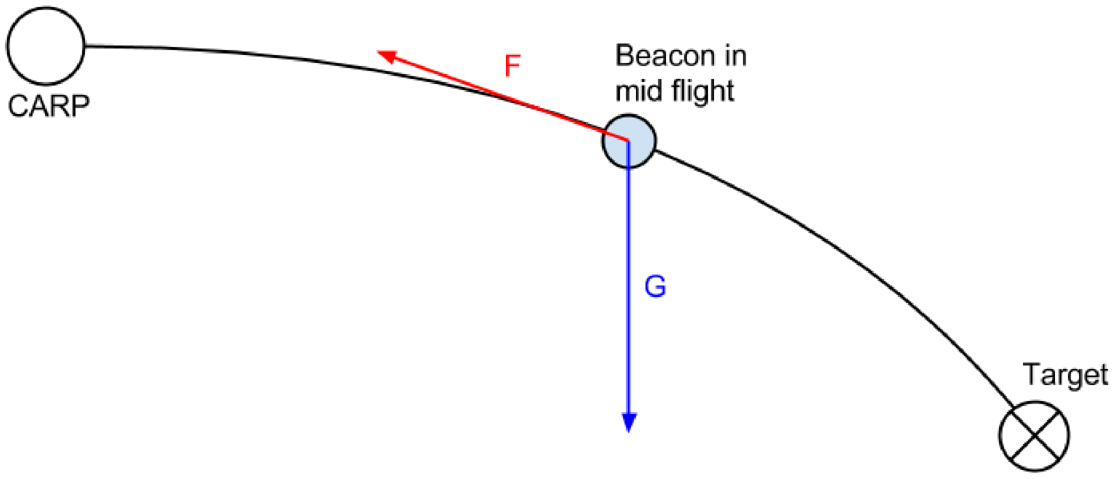


Figure 2.1: The ballistic path of the beacon, F signifies air resistance and G gravitational pull

2.1 Aerial Release Point

The CARP depends on the ballistics of the payload during the drop given wind and initial conditions of the UAV and payload.

2.1.1 Drag

As the beacon is dropped from the UAV, it will follow a ballistic path caused by gravity and drag as seen in Figure 2.1. This trajectory will be defined by the properties of the payload and its initial conditions, drag and gravity. The drag coefficient plays a significant role in defining the ballistics of the beacon (Sherwood, 1967). The drag force, F_d, acting on the payload is usually expressed using the formula Lord Rayleigh first devised

$$F_d := \frac{1}{2} C_d A \rho \bar{v}^2, \qquad \bar{v} > 0 \tag{2.1}$$

where A is the projected area, C_d the dimensionless drag coefficient and $\bar{v}$ the relative velocity with regards to the fluid medium and its density, ρ.

When velocity is given as a vector in three dimensions, the drag force is then expressed as

$$\mathbf{F_d} := \frac{1}{2} C_d A \rho \bar{\mathbf{v}} \|\bar{\mathbf{v}}\|, \qquad \|\bar{\mathbf{v}}\| := \sqrt{\bar{v}_x^2 + \bar{v}_y^2 + \bar{v}_z^2} > 0, \qquad \mathbf{F_d} := \begin{pmatrix} F_{d_x} \\ F_{d_y} \\ F_{d_z} \end{pmatrix}, \qquad \bar{\mathbf{v}} := \begin{pmatrix} \bar{v}_x \\ \bar{v}_y \\ \bar{v}_z \end{pmatrix} \tag{2.2}$$

where $\bar{v}_x$, $\bar{v}_y$ and $\bar{v}_z$ are the decomposed velocities with respect to the fluid and F_{d_x}, F_{d_y} and F_{d_z} are the decomposed drag forces in x, y and z, respectively.

The drag coefficient is a function of the Reynolds number only, $R_e = D\bar{v}/v$, where D is the object dimension and v is the kinematic viscosity of the fluid medium (Sherwood, 1967). However, since the inertial drag dominates in our case, which is signified by a large R_e, we can assume C_d constant. The magnitude of C_d then depends on the shape of the object. For $R_e \geq 10^3$, which is a good assumption for the purposes of this project, C_d is normally between 0.3 and 1.5. (Sherwood, 1967).

2.1.2 Payload Equations of Motion

Using Newton's second law to find the decomposed equations of motion (EoM) for the beacon, and assuming that the beacon is a point mass and that the only forces acting on the beacon is gravity and air resistance, this becomes the following EoM for the xz-plane

$$m\dot{v}_z = -mg - F_{d_z} \tag{2.3}$$

$$m\dot{v}_x = -F_{d_x} \tag{2.4}$$

where gravity points in negative-z direction, g is the gravitation constant, v_x and v_z are the decomposed velocities in x and z direction with respect to the ground, and m is the mass of the beacon, by combining 2.2 and 2.3 and dividing by m

$$\dot{v_z} = -g - \frac{b}{m}\bar{v_z}\|\bar{\mathbf{v}}\| \tag{2.5}$$

$$\dot{v_x} = -\frac{b}{m}\bar{v_x}\|\bar{\mathbf{v}}\| \tag{2.6}$$

where $b := \frac{1}{2}C_d\rho A$. The relative velocities affecting the UAV can be found by using the UAV's speed and wind estimates, $\mathbf{v}$ and $\mathbf{w}$. Substituting this into 2.5 and 2.6 will result in

$$\dot{v_z} = -g - \frac{b}{m}(v_z - w_z)\|\mathbf{v} - \mathbf{w}\| \tag{2.7}$$

$$\dot{v_x} = -\frac{b}{m}(v_x - w_x)\|\mathbf{v} - \mathbf{w}\| \tag{2.8}$$

where

$$\mathbf{v} := \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \qquad \mathbf{w} := \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} \tag{2.9}$$

where v_y is the decomposed velocity with respect to ground. Furthermore, calculating the impact point of the drop can then be done by integration of v_x and v_z. Extending the EoM to y direction as well is trivial, since the EoM in y-direction are the same as in the x-direction. The differential equations of the system are then

$$\begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -\frac{b}{m}(v_x - w_x)\|\mathbf{v} - \mathbf{w}\| \\ -\frac{b}{m}(v_y - w_y)\|\mathbf{v} - \mathbf{w}\| \\ -g - \frac{b}{m}(v_z - w_z)\|\mathbf{v} - \mathbf{w}\| \\ v_x \\ v_y \\ v_z \end{bmatrix} \tag{2.10}$$

These six ordinary differential equations will give the ballistic trajectory by inputting certain initial conditions. Giving a velocity and start position [0, 0, release height] as input and looking

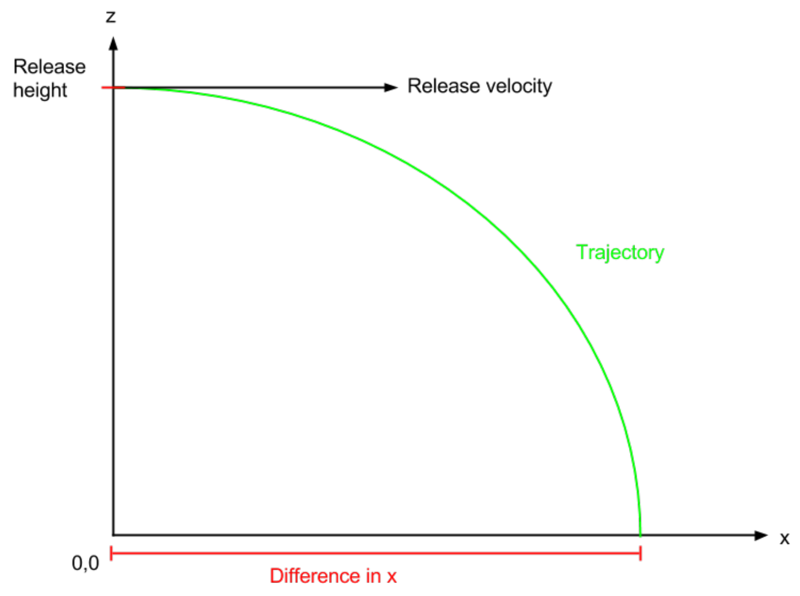


Figure 2.2: Trajectory visualising the distance travelled in x

at the trajectory until $z = 0$, the EoM will give a trajectory with a known start point and an end point from which you can read the distance travelled in x and y as the x and y coordinate (the difference in x and y). If the target coordinate is displaced in the xy-plane by minus the distance travelled in x and y , and displaced in z by minus the release height, the end position is at the target and the start position is the displaced point. The displaced point is then the CARP, as visualized in Figure 2.1 and Figure 2.2.

2.1.3 Feasible Aerial Release Points

Let the feasible aerial release points (FARP) be the set of all possible CARPs. It is then defined by all initial states in (2.10) such that somewhere along the trajectory, the beacon's coordinates coincides with the target's coordinates.

Some simplifications are used in this thesis.

- The initial v_z is zero
- The release height is the same for all CARPs
- w_z is zero
- The speed upon release is the same for all CARPs ($\sqrt{v_{x_{initial}}^2 + v_{y_{initial}}^2} = V_{total}$, where V_{total} is a constant)

Expressed more intuitively, the height and speed are assumed constant when the beacon is released. This leads to the FARP being a line around the target where every choice of $(x_{initial}, y_{initial})$ has a corresponding $(v_{x_{initial}}, v_{y_{initial}})$. If an analytical solution was found for this line, it could be used in a typical optimization problem. The analytical solution was not found, and so an iterative method was used instead. The iterative method is described in Section 2.1.4 and the results are used in Section 2.5.3.

2.1.4 Iterative Calculation of the Aerial Release Point

Euler's method (Egeland and Gravdahl, 2003, p.521-523) was chosen to calculate the trajectory from the initial point until ground level is reached.

Implementation

This method calculates a trajectory for equation 2.10. Many other iterative methods are available, but Euler's method was chosen as it is simple to implement. The method works by adding the current value and its derivative times a step length to get the next value. Used on Equation 2.10, the iteration becomes

$$\begin{bmatrix} v_{x_{n+1}} \\ v_{y_{n+1}} \\ v_{z_{n+1}} \\ x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{bmatrix} = \begin{bmatrix} v_{x_n} - \frac{b}{m}(v_{x_n} - w_{x_n})\|\mathbf{v}_n - \mathbf{w}_n\|\delta_t \\ v_{y_n} - \frac{b}{m}(v_{y_n} - w_{y_n})\|\mathbf{v}_n - \mathbf{w}_n\|\delta_t \\ v_{z_n} - (g + \frac{b}{m}(v_{z_n} - v_{z_n})\|\mathbf{v}_n - \mathbf{w}_n\|)\delta_t \\ x_n + v_{x_n}\delta_t \\ y_n + v_{y_n}\delta_t \\ z_n + v_{z_n}\delta_t \end{bmatrix} \tag{2.11}$$

Where δ_t is the step length and n signifies step number. For this method to work with the objectives in this thesis, the initial conditions must be set such that the trajectory in the xy plane becomes apparent. The release height, wind, drag coefficients (b), step size and velocity are given as input to the iterative program. Seen in the xyz plane, the start point is in [0,0,h], where h is release height. The iteration then starts and runs until the z value becomes less than, or equal to, zero. The values of x and y then describes the movement in north and east for that trajectory. It is then straight-forward to displace the target coordinates to find the CARP.

Stability Properties of the Iterative Method

Looking at the stability properties of Euler's method for this system using example 214 page 522 in Egeland and Gravdahl (2003)

$$\dot{v}_{xy} = -\frac{b}{m}(v_{xy} - w_{xy})\|\mathbf{v} - \mathbf{w}\| \tag{2.12}$$

$$\dot{v}_z = g - \frac{b}{m}(v_z - w_z)\|\mathbf{v} - \mathbf{w}\| \tag{2.13}$$

where v_{xy} is velocity in x or y directions as their equations are the same. It is then approximated that $\|\mathbf{v} - \mathbf{w}\|$ is the largest possible velocity, the terminal velocity, v_t

$$\dot{v}_{xy} = -\frac{b}{m}(v_{xy} - w_{xy})v_t \tag{2.14}$$

$$\dot{v}_z = g - \frac{b}{m}(v_z - w_z)v_t \tag{2.15}$$

$$\frac{d\dot{v}_z}{dv_z} = \frac{d\dot{v}_{xy}}{dv_{xy}} = -\frac{b}{m}v_t \tag{2.16}$$

As the derivatives of $\dot{v}_{xy}$ and $\dot{v}_z$ are the same, only one equation is used for all directions. Linearising gives

$$\Delta\dot{v}_{xyz} = -\frac{b}{m}v_t\Delta v_{xyz} \tag{2.17}$$

where the v_{xyz} is velocity in all directions. This gives the largest step limit

$$h \leq \frac{2}{\frac{b}{m}v_t} \tag{2.18}$$

where h is the step limit with the terminal-velocity approximation. The step limit goes down with higher velocities, and as the approximation uses the highest speed possible, the step is as small as is required by any possible speed. Calculation of v_t follows by Newton's second law and that terminal velocity is achieved when G equals F_{drag} and acceleration is zero.

$$ma = \Sigma F \tag{2.19}$$

$$ma = G - F_{drag_{max}} = 0 \tag{2.20}$$

$$mg = F_{drag_{max}} \tag{2.21}$$

$$mg = bv_t^2 \tag{2.22}$$

$$v_t = \sqrt{\frac{mg}{b}} = 41.74[m/s] \tag{2.23}$$

where $F_{drag_{max}}$ is the maximum drag force, g is 9.81 m/s^2 and b and m are given in 3.1.9. Inserting b and m, from Section 3.1.9, and v_t, into 2.18 finally leads to

$$h \leq 8.51[s] \tag{2.24}$$

Another condition to take into account when discussing step length is the detection of crossing the plane at $z = 0$. With the constant terminal velocity in mind, distance travelled in one step is

$$s = hv_t \tag{2.25}$$

where s is the distance travelled in one step. This distance is equal to the CARP calculation accuracy in z direction. If the needed accuracy is 0.1 m, the step length becomes

$$h = \frac{s}{v_t} = 0.00240[s] \tag{2.26}$$

As the time step used for this thesis' experiments is 0.001 s, the accuracy is

$$s = hv_t = 0.042[m] \tag{2.27}$$

2.2 UAV Reference Frames and Kinematics

To be able to control the UAV, its reference frames and kinematics must be described. As this is fairly common theory, this thesis tries to keep to industry standard. In the rest of this thesis, several controllers and guidance systems will rely on these.

First, we need to introduce some relevant notation and definitions such as reference frames

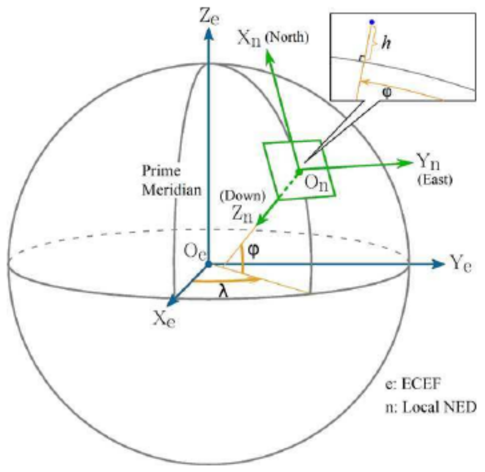


Figure 2.3: The Geodetic, ECEF, and local NED coordinate systems (Brezoescu, 2014)

(Figure 2.3) and mathematical notation, then the kinetics are described in component form.

2.2.1 Reference frames

Geodetic coordinate system: The geodetic coordinate system is a widely used in GPS-based navigation. This coordinate system as defined near the earth surface in terms of the longitude, latitude and height, denoted (l, μ, h) respectively.

ECI: Earth-centred inertial reference frame $\{i\} = (x_i, y_i, z_i)$ is a global coordinated system with the origin O_i in the center of the earth. This is a non-accelerating fixed reference frame with its origin in the center of the earth.

ECEF: Earth-centred earth fixed reference frame $\{e\} = (x_e, y_e, z_e)$ is a global coordinated system with the origin $O_e = O_i$ in the center of the earth. This frame is is rotating relative to the ECI, where the origin of the ECEF is the same as for ECI. For a vehicle moving at relative low speeds, this frame can be seen as inertial.

Local NED: Local north-east-down reference frame $\{n\} = (x_n, y_n, z_n)$ is a local reference frame with origin O_n, often used to get better resolution of the area of interest. Its coordinate frame is fixed to the earth's surface, based on the World Geodetic System 84 (WGS84) ellipsoid. The z-axis points downwards perpendicular to the plane tangent to the ellipsoid, the x-axis points towards true north and the y-axis points towards east. Guidance and navigation are normally carried out in this frame.The location $\{n\}$ relative to $\{e\}$ determined by the longitude and latitude. For a vehicle operating in a local area, approximately constant longitude and latitude, can we assume $\{n\}$ as inertial, so that Newton's laws applies.

Vehicle carried NED: Vehicle carried north-east-down reference frame $\{nv\} = (x_{nv}, y_{nv}, z_{nv})$ is a local reference frame with origin O_{nv}. This reference frame is associated with the flying vehicle. The axis directions of the vehicle-carried NED frame vary with respect to the flying-vehicle movement and are thus not aligned with those of the local NED frame. However, miniature UAVs fly only in a small region with low speed, which results in the directional difference being completely neglectable. It is thereby reasonable to assume that the directions of the vehicle-carried NED constantly coincide with the local NED coordinate system.

Body: The body reference frame $\{b\} = (x^b, y^b, z^b)$ is a local reference frame that is fixed to the vehicle at a predefined point on the vehicle. Where axes are usually chosen so that x_b points in the forward direction, y_b points to the right side and z_b points downward.

Wind (and stability): In addition to body reference frame, is it common to include the wind $\{w\}$ and stability $\{s\}$ coordinate frames for aircraft. These frames has it origin in the aircraft center of gravity Figure 2.4.

2.2.2 Coordinate transformation

Some necessary and relevant coordinate transformation need to be included (Brezoescu, 2014).

Geodetic and ECEF:. The positions given by the GPS are most often provided as ellipsoidal coordinates (latitude, longitude and height) based on the World Geodetic System 84 (WGS84) ellipsoid. In order to convert the GPS measurements to the local NED to we first need an inter-

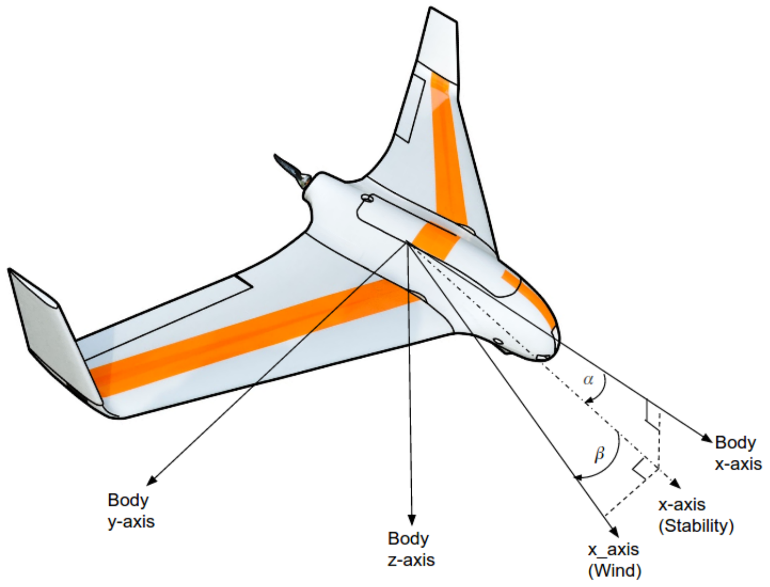


Figure 2.4: Definition of stability and wind axis frames. Where $\alpha = w/u$ is the angle of attack and $\beta = v/v_T$ is the sideslip angle

mediate step converting GPS to ECEF. This can be done in the following way

$$\mathbf{p}^e_{b/e} = \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} = \begin{bmatrix} (N_E + h)cos(\mu)cos(l) \\ (N_E + h)cos(\mu)sin(l) \\ [N_E(1-e^2)+h]sin(\mu) \end{bmatrix} \tag{2.28}$$

where e is the eccentricity of the ellipsoid and $N_E = r_e^2/\sqrt{r_e^2 cos^s(\mu) + r_p^2 sin^2(\mu)}$ is the prime vertical radius of the curvature, where r_e is the semi-major axis and r_p is the semi-minor axis of the ellipsoid.

ECEF and Local NED:. The position transformation from local NED to ECEF are defined as follows

$$\mathbf{v}^e_{b/e} = \dot{\mathbf{p}}^e_{b/e} = \mathbf{R}^e_n(\Theta_{ne})\dot{\mathbf{p}}^n_{b/e} \tag{2.29}$$

where $\mathbf{p}^n_{b/e}$ is the position of the point O_b with respect to the $\{e\}$ frame represented in the $\{b\}$

frame, $\Theta_{ne} = [l, \mu]^T$ and the rotation matrix between ECEF to local NED are

$$\mathbf{R}_n^e(\Theta_{ne}) = \begin{bmatrix} -cos(l)sin(\mu) & -sin(\mu) & -cos(l)cos(\mu) \\ -sin(l)sin(\mu) & -cos(\mu) & -sin(l)cos(\mu) \\ cos(\mu) & 0 & -sin(\mu) \end{bmatrix} \tag{2.30}$$

and $\mathbf{p}_{b/e}^e$ can be found by integrating 2.29.

Geodetic and Vehicle Carried NED: The relationship between geodetic position and vehicle carried NED, where the geodetic position and the vehicle carried NED velocity are of great interest.

$$\begin{bmatrix} \dot{l} \\ \dot{\mu} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} \frac{v_{nv}}{(N_E+h)cos(\mu)} \\ \frac{u_{nv}}{M_E+h} \\ -w_{nv} \end{bmatrix} \tag{2.31}$$

The derivatives of the vehicle carried NED are then given as

$$\mathbf{V}_{b/nv}^{nv} = \begin{bmatrix} \dot{u}_{nv} \\ \dot{v}_{nv} \\ \dot{w}_{nv} \end{bmatrix} = \begin{bmatrix} -\frac{v_{nv}^2 sin(\mu)}{(N_E+h)cos(\mu)} + \frac{u_{nv}w_{nv}}{M_E+h} + a_{mx_{nv}} \\ \frac{u_{nv}v_{nv}sin(\mu)}{(N_E+h)cos(\mu)} + \frac{u_{nv}w_{nv}}{N_E+h} + a_{my_{nv}} \\ -\frac{v_{nv}^2}{(N_E+h)} - \frac{u_{nv}^2}{M_E+h} + g + a_{mz_{nv}} \end{bmatrix} \tag{2.32}$$

where g is the gravitational constant and $\mathbf{a}^{nv} = [a_{mx_{nv}}, a_{my_{nv}}, a_{mz_{nv}}]^T$ is the projection of $\mathbf{a}^b = [a_{mx_b}, a_{my_b}, a_{mz_b}]^T$ (the acceleration measured on the body) onto the vehicle carried NED. The acceleration $\mathbf{a}^b$ can typically be measured by a three-axis accelerometer, assuming that the center of origin (CO) of the accelerometer coincide with the center of gravity of the vehicle. This will be assumed in this report. However, in the case where accelerometer origin does not coincide with the CG, transformation is needed.

Vehicle Carried NED and Body: The kinematic relation between the vehicle carried NED and body, this relationship is important in modelling and control of a vehicle.

$$\mathbf{v}^{b}_{b/nv} = \mathbf{R}^{b}_{nv}(\Theta_{nvb})\mathbf{v}^{nv}_{b/nv} \tag{2.33}$$

$$\mathbf{a}^{b}_{b/nv} = \mathbf{R}^{b}_{nv}(\Theta_{nvb})\mathbf{a}^{nv}_{b/nv} \tag{2.34}$$

where $\mathbf{v}^{b}_{b/nv} = [u, v, w]^T$, $\mathbf{a}^{b}_{b/nv} = [a_x, a_y, a_z]^T$, $\Theta_{nvb} = [\phi, \theta, \psi]^T$ and $\mathbf{R}^{b}_{nv}(\Theta_{nvb})$ is the euler angel rotation matrix defined as

$$\mathbf{R}^{b}_{nv}(\Theta_{nvb}) = \begin{bmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & sin(\theta) \\ sin(\phi)sin(\theta)cos(\psi) - cos(\phi)sin(\psi) & sin(\phi)sin(\theta)cos(\psi) + cos(\phi)cos(\psi) & sin(\phi)cos(\theta) \\ cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) & cos(\phi)cos(\theta) \end{bmatrix} \tag{2.35}$$

Local and vehicle carried NED: Assuming that the UAV only flies in a small region at low speeds as described in 2.2.1, can we assume that there is no difference between $\{nv\} = \{n\}$. This is a good assumption in our case, and will be followed throughout the report.

Wind and body: The relation between $\{w\}$ and $\{b\}$ frame is determined by the aerodynamic angles, seen in Figure 2.4, β and α, which stand for sideslip and angle of attack, respectively.

$$\mathbf{v}^{w}_{b/n} = \mathbf{R}^{w}_{b}(\Theta_{bw})\mathbf{v}^{b}_{b/n} \tag{2.36}$$

where $\mathbf{v}^{b}_{b/n}$ is the airmass velocity with respect to the $\{n\}$ expressed in $\{b\}$, $\mathbf{v}^{w}_{b/n}$ is the airmass velocity with respect to the $\{n\}$ expressed in $\{w\}$ and $\Theta_{bw} = [-\beta, \alpha]^T$. Furthermore

$$R^{w}_{b}(\Theta_{bw}) = \begin{bmatrix} cos(\alpha)cos(\beta) & sin(\beta) & sin(\alpha)cos(\beta) \\ -sin(\alpha)cos(\beta) & cos(\beta) & -sin(\alpha)sin(\beta) \\ -sin(\alpha) & 0 & cos(\alpha) \end{bmatrix} \tag{2.37}$$

2.2.3 Kinematics and Kinetics

Assuming that vehicle (UAV) is rigid and that the local NED frame is inertial, the kinematic equations of motion for the UAV can be described as follows (Figure 2.5) (Langelaan et al., 2011). The vector relationship between this $\vec{\mathbf{v}}_g = \vec{\mathbf{v}}_a + \vec{\mathbf{w}}$. Using the vector notation from Fossen (2011) this becomes

$$\dot{\mathbf{p}}^n_{g/e} = \mathbf{v}^n_{g/e} = \mathbf{R}^b_n(\Theta_{nb})^{-1}\mathbf{v}^b_{a/e} + \mathbf{v}^n_{w/e} \tag{2.38}$$

where $\mathbf{v}^n_{w/e} = [w_{x_e}, w_{y_e}, w_{z_e}]^T$ is the wind speed components with respect to the $\{e\}$ expressed in $\{n\}$ and $\mathbf{v}^n_{a/e} = [u, v, w]^T$ is the relative aircraft velocity with respect to $\{e\}$ expressed in $\{n\}$. Furthermore the rigid body kinetics for the velocity can be described in component from as follows (Langelaan et al., 2011)

$$\dot{u} = \frac{X}{m} - gsin(\theta) - qw + rw - \dot{w}_{x_e}cos(\theta)cos(\psi) - \dot{w}_{y_e}cos(\theta)sin(\psi) + \dot{w}_{z_e}sin(\theta) \tag{2.39}$$

$$\begin{aligned}\dot{v} = &\frac{Y}{m} + gsin(\phi)cos(\theta) + pw - ru - \dot{w}_{x_e}(sin(\phi)sin(\theta)cos(\psi) - cos(\phi)sin(psi)) \\ &- \dot{w}_{y_e}(sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(psi)) - \dot{w}_{z_e}sin(\phi)cos(\theta)\end{aligned} \tag{2.40}$$

$$\begin{aligned}\dot{w} = &\frac{Z}{m} + gcos(\phi)cos(\theta) + qu - pv - \dot{w}_{x_e}(cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(psi)) \\ &- \dot{w}_{y_e}(cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(psi)) - \dot{w}_{z_e}cos(\phi)cos(\theta)\end{aligned} \tag{2.41}$$

2.3 Wind Estimation

Using GPS measured velocity, a pitot tube and IMU measurements, we can derive a method for directly computing the wind velocity with respect to the earth (Johansen et al., 2015; Langelaan et al., 2011). A differential GPS provides a direct measurement of the ground velocity with respect to the earth $\dot{\mathbf{p}}^e_{g/e} = \mathbf{v}^e_{g/e}$, this is obtained by using the rates of change of pseudoranges. The pseudrorange is the pseudo distance between a satellite and a navigation satellite receiver (GPS) (Enge, 1994).

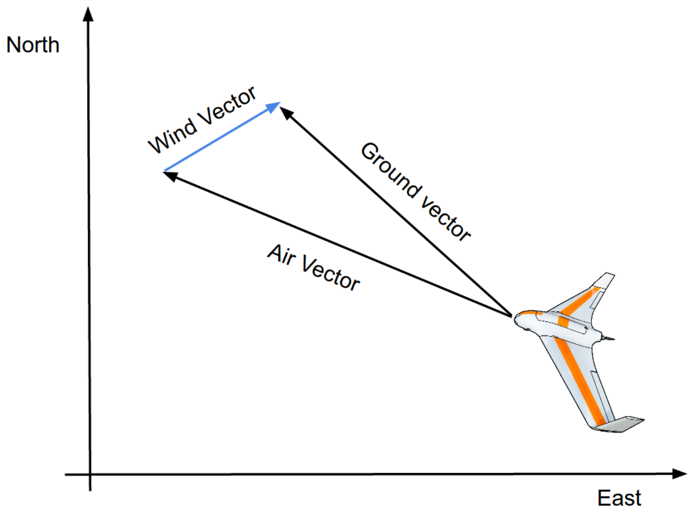


Figure 2.5: Graphical vector relationship between aircraft motion and wind, represented in two-dimensional (north and east) local NED frame

Additionally assuming that the autopilot module provides estimates of the UAV's air speed and attitude means that the local wind velocity components can be obtained directly from the UAV's attitude, ground speed and airspeed (Langelaan et al., 2011).

$$\begin{aligned}\mathbf{v}^n_{w/e} &= \mathbf{v}^n_{g/e} - \mathbf{R}^b_n(\Theta_{nb})^{-1}\mathbf{v}^b_{a/e} \\ \mathbf{v}^n_{w/e} &= \mathbf{v}^n_{g/e} - \mathbf{R}^b_n(\Theta_{nb})^{-1}\mathbf{R}^w_b(\Theta_{bw})^{-1}\mathbf{v}^w_{a/e} \\ &= \mathbf{R}^e_n(\Theta_{ne})^{-1}\mathbf{v}^e_{g/e} - \mathbf{R}^b_n(\Theta_{\mathbf{nb}})^{-1}\mathbf{R}^w_b(\Theta_{bw})^{-1}\mathbf{v}^w_{a/e}\end{aligned} \tag{2.42}$$

where $\mathbf{v}^w_{a/n} = [v_a, 0, 0]^T$, $v_a = \sqrt{u^2 + v^2 + w^2}$ is the total airspeed, where u, v and w is the velocity in body x, y and z direction, respectively.

The estimator proposed in Johansen et al. (2015) was implemented to find the wind velocity. It is based on an observer using a Kalman filter.

The measured airspeed is modelled as $u^m_r = \gamma u_r$ and is measured by the airspeed sensor, in this case a pitot tube. γ is a scaling factor for online calibration, hence the observer injection term is given as

$$u = d_1^T \mathbf{R}_n^b \mathbf{v}_w^n + u_r \gamma \tag{2.43}$$

where $d_1 = [1, 0, 0]^T$ and $\mathbf{R}_n^b$ is the body-to-ned Euler rotation matrix. Both $\mathbf{v}_w$ and γ are considered slowly time varying ($\dot{\mathbf{v}}_w = \dot{\gamma} = 0$). This gives system matrices

$$\mathbf{A} = 0 \tag{2.44}$$

$$\mathbf{C} = \begin{bmatrix} d_1^T \mathbf{R}_n^b & u_r \end{bmatrix} \tag{2.45}$$

$$\mathbf{D} = d_1 d_1^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.46}$$

Due to the rank of $\mathbf{D}$ being 1, observability must be acquired through manipulation of $\mathbf{R}_n^b$. This is managed by varying the attitude through manoeuvres at all times to keep the observability gramian positive for all time intervals. More information on this can be found in Johansen et al. (2015).

The observer is

$$\begin{bmatrix} \dot{\hat{\mathbf{v}}}_w^n \\ \dot{\hat{\gamma}} \end{bmatrix} = \mathbf{K}(\hat{u} - \hat{\gamma} u_r^m - d_1^T \hat{\mathbf{R}}_n^b \hat{\mathbf{v}}_w^n) \tag{2.47}$$

Which is then processed in a Kalman-Bucy filter

$$\mathbf{K} = \mathbf{P}\mathbf{C}^T \mathbf{R}^{-1} \tag{2.48}$$

$$\dot{\mathbf{P}} = \mathbf{Q} - \mathbf{K}\mathbf{R}\mathbf{K}^T \tag{2.49}$$

where $\dot{\hat{\mathbf{v}}}_w^n$, $\dot{\hat{\gamma}}$, $\hat{u}$, $\hat{\mathbf{v}}_w^n$, $\hat{\gamma}$ and $\hat{\mathbf{R}}_n^b$ are estimated entities, $\mathbf{R}$ is the variance of the pitot tube measurement noise, $\mathbf{Q} > 0$ is the covariance of a white noise model of the wind velocity and scaling factor change and the initial condition for $\mathbf{P}$ is $\mathbf{P}(\mathbf{0}) > 0$. Both $\mathbf{P}$ and $\mathbf{Q}$ are symmetric matrices. As the PE condition is satisfied, $\mathbf{P}$ is bounded and the observer is globally exponentially stable.

This would give a fairly good estimate of the local wind speed. However, GPS velocity noise effects and the accuracy of the estimates of both airspeed and Euler angles will affect the result. Note that the wind estimates found by this method are assuming uniform wind components in the local area.

2.4 Flight Controllers

To control the UAV, lateral controllers are needed. These controllers will help guide the UAV to the CARP. In this thesis, two guidance controllers using two course controllers have been tested. The course controllers are based on a bank to turn controller, as can be seen in Figure 2.6.

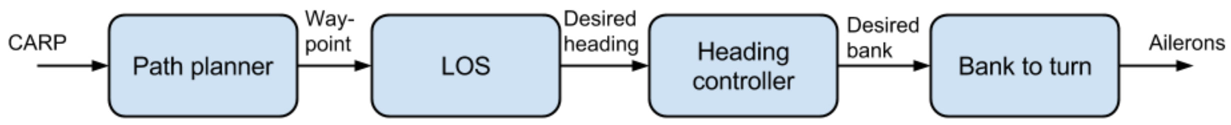


Figure 2.6: Controller hierarchy

2.4.1 Guidance Controllers

Two guidance controllers have been tested, the L1 controller and a line-of-sight (LOS) controller.

L1 Controller

The simplest way to control the UAV is to use the Pixhawk's built in guidance controller. To use it, a waypoint coordinate is sent to the Pixhawk through one of its input ports. The controller can be used in two ways, either as a pseudo LOS controller (then referred to as the L1 controller) or as a waypoint controller (then referred to as the waypoint controller). They are described as Region B and Region A in drawings from Plane: L1 Control (2013). The L1 controller was not tested in this thesis, as its implementation with DUNE (Section 4.1.3) was not yet completed.

When used as a pseudo LOS, it is based on Park et al. (2004). The control law is then

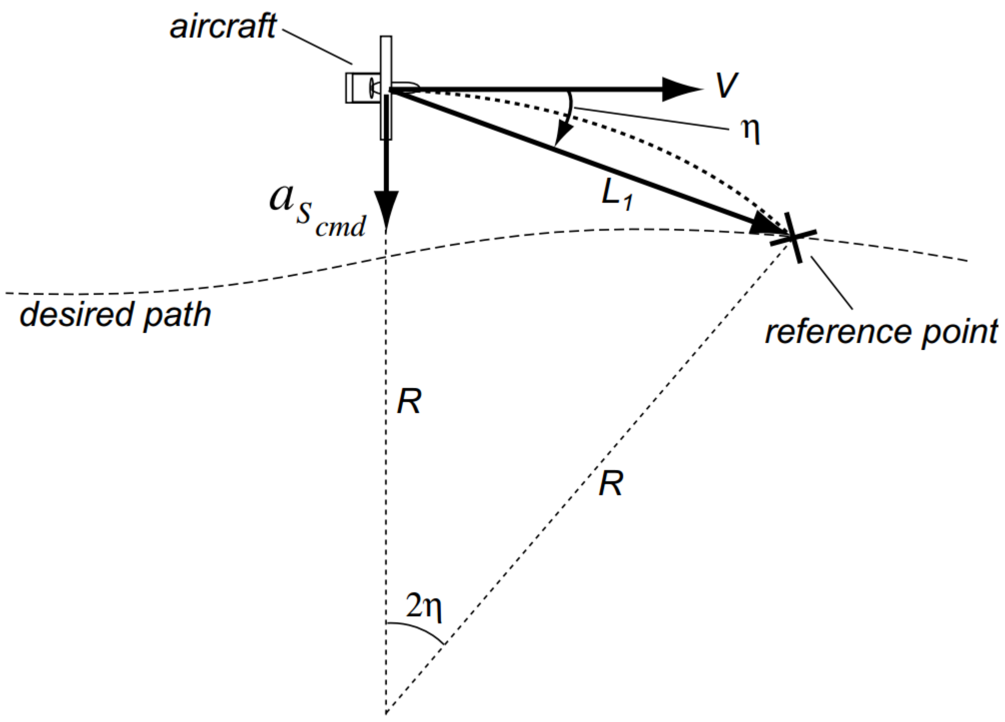


Figure 2.7: L1 guidance

$$\Delta\Psi = \frac{a_s}{V}\Delta t \tag{2.50}$$

$$a_s = 2\frac{V^2}{L_1}\sin\eta \tag{2.51}$$

where $\Delta\Psi$ is the desired change in heading, a_s is the desired centripetal acceleration, V is the UAV ground velocity, Δt is the controller step time, L_1 is the look ahead distance for this controller and η is the error in heading. These are visualized in Figure 2.7.

When used as a waypoint controller, the autopilot then tries to go towards the waypoint by setting the desired course towards it in relation to the current UAV position. That is to say, the controller compares the UAV's position to the CARP and defines a desired course (χ_d). When the UAV reaches the final waypoint, it will start loitering around the coordinate with a given radius. This is a good safety measure as the UAV will never stop, but reduces accuracy when getting close to the loitering circle as the UAV will start turning into the circular pattern.

Line-of-Sight Controller

To answer the waypoint controller's downsides, a LOS controller can be used instead. This controller uses a start point and an end point to calculate a straight line which it will try to follow. The CARP will be reached with the correct velocity if the line is placed on top of the CARP in the same direction as the CARP's corresponding velocity. The accuracy of this method depends on the tuning of the LOS controller and the underlying bank-to-turn and course controller.

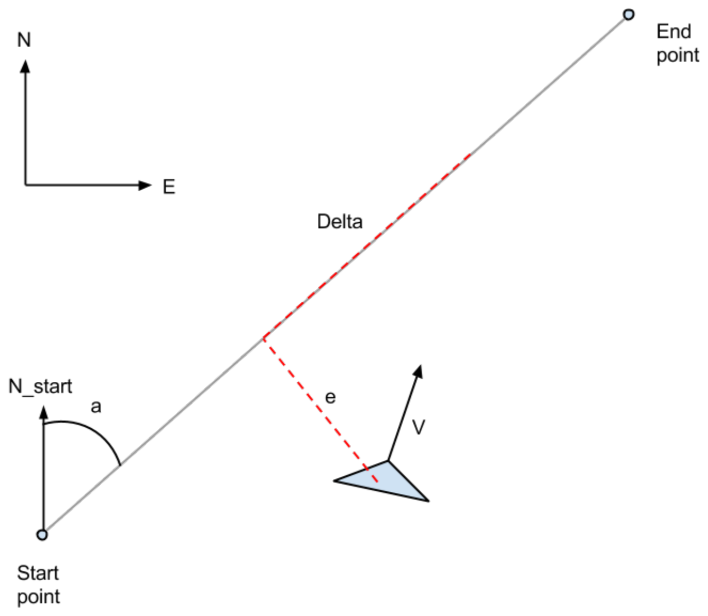


Figure 2.8: LOS guidance

LOS guidance is based on the following formulae from Fossen (2011)

$$\chi_d(e) = \chi_p + \chi_r(e) \tag{2.52}$$

$$\chi_p = a \tag{2.53}$$

$$\chi_r(e) := atan\frac{-e(t)}{\Delta} \tag{2.54}$$

$$\begin{bmatrix} s(t) \\ e(t) \end{bmatrix} = \begin{bmatrix} cos(\alpha_k) & -sin(\alpha_k) \\ sin(\alpha_k) & cos(\alpha_k) \end{bmatrix} \left(\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} - \begin{bmatrix} x_{start} \\ y_{start} \end{bmatrix} \right) \tag{2.55}$$

assuming sideslip angle $\beta = 0$, where χ_d is desired course, χ_p is start point to end point course and χ_r is reference course based on path following error, $e(t)$. $s(t)$ is the along path distance travelled, $x(t)$ is UAV north coordinate, $y(t)$ is the UAV east coordinate, x_{start} is the north coordinate of the path start point, y_{start} is the east coordinate of the path start point and Δ is the look ahead distance. Some of these variables are found in Figure 2.8.

2.4.2 Heading Controllers

As seen in Section 2.4.1, the guidance controllers use a desired course, χ_d, or change in heading, $\Delta\Psi$, to follow the path. To achieve this, a heading or a course controller is implemented. The difference between the two is the side slip affecting the UAV.

Bank Angle Translation

As the L1 controller outputs a desired $\Delta\Psi$, this can be inserted into

$$\dot{\psi} = \frac{g}{V_a}\tan\phi \tag{2.56}$$

from Beard and McLain (2012), where $\dot{\psi}$ is approximately $\Delta\Psi$, V_a is the air speed and ϕ is the desired roll angle. The desired roll angle is then input to the Pixhawk's roll angle controller, which is a part of the bank-to-turn controller.

Sliding Mode Controller

A controller used to achieve $\chi_d(e)$ can be found in Fortuna and Fossen. It is a sliding mode controller (SMC) with uniform semi-global exponential stability. It is a course controller as opposed to a heading controller, and will therefore account for side slip angle. Tuning is also taken directly from the paper. This controller's output is the desired roll.

$$\phi_d = (-\lambda\dot{\tilde{\chi}} - \rho\,\mathrm{sgn}(s) - K_d s)\frac{V_g^2 T_\phi \epsilon^2(\phi)}{g(V_a + W_x\cos(\psi) + W_y\sin(\psi))} \tag{2.57}$$

where ϕ_d is the desired roll, λ, ρ and K_d are tuning parameters, $\rho \geq f_{max} \geq |\ddot{\chi}|_{input=0} + |\ddot{\chi}_d|$, s is the sliding mode surface, $s = \dot{\tilde{\chi}} + \lambda\tilde{\chi}$, T_ϕ is the time constant for the roll angle, ϵ is $\cos\phi$ rewritten to avoid singularity (see Fortuna and Fossen) and W_x and W_y are wind velocities. The roll angle is controlled by the Pixhawk's internal bank controller.

2.5 Path Planners

To drop the beacon accurately, the chosen CARP must be reached with the correct velocity. Several paths to reach the CARP has been developed and explored in this thesis. They are described in the following sections. All of them use a glide state before dropping to avoid interaction between propeller and beacon.

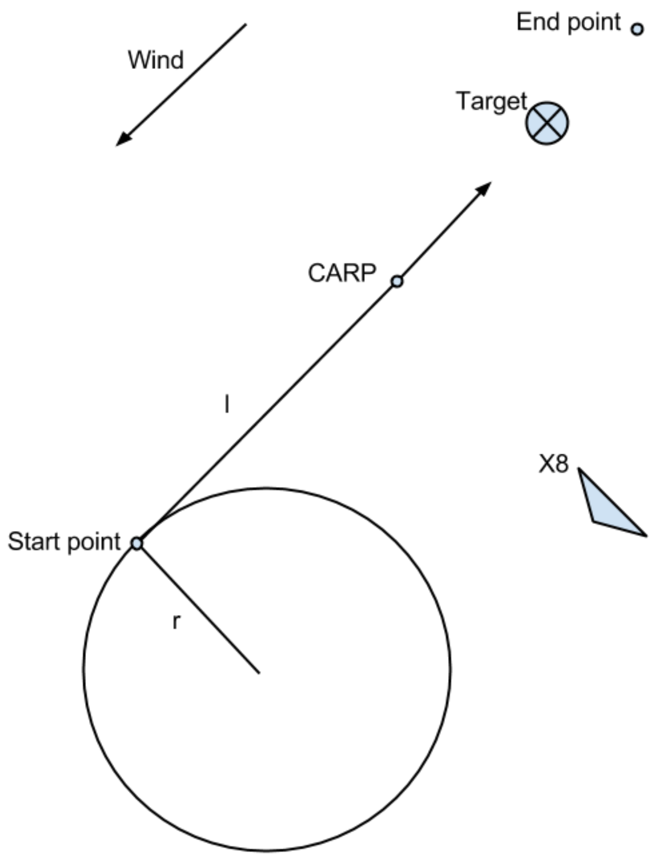


Figure 2.9: The Straight-Line path

2.5.1 Straight-Line

The Straight-Line (SL) path is a simplified version of the Adapted Dubins Path algorithm from Fuglaas (2014). It is based on making the UAV go to a start point and then to an end point directly opposite the CARP. The start and end point is chosen so that the path is up against the wind, and in the same direction as the CARP velocity. The velocity direction of the UAV in the start position is toward the end position. If this is acheived, the UAV should in theory reach the CARP with the appropriate velocity.

To set up the line during testing, a waypoint is chosen l meters away from the CARP and r meters to the side (where r is the loitering radius). When the UAV is loitering and pointing towards the CARP (within a certain accuracy), it is given the end point as the next waypoint to go to, and it will then go towards the CARP. This is shown in Figure 2.9.

The CARP is chosen based on the wind direction. During CARP calculations, the UAV's velocity vector is set opposite to the wind current to reduce the side slip while approaching the CARP. This also leads to a CARP between the start point and target, as close to the target as possible.

In theory, both the waypoint controller and LOS controller from Section 2.4.1 should work with this path planner, but it is more robust while using LOS as it can compensate for disturbances pushing it away from the line, while the waypoint controller does not.

A disadvantage of this path planner is that it does not update the CARP if the wind has changed.

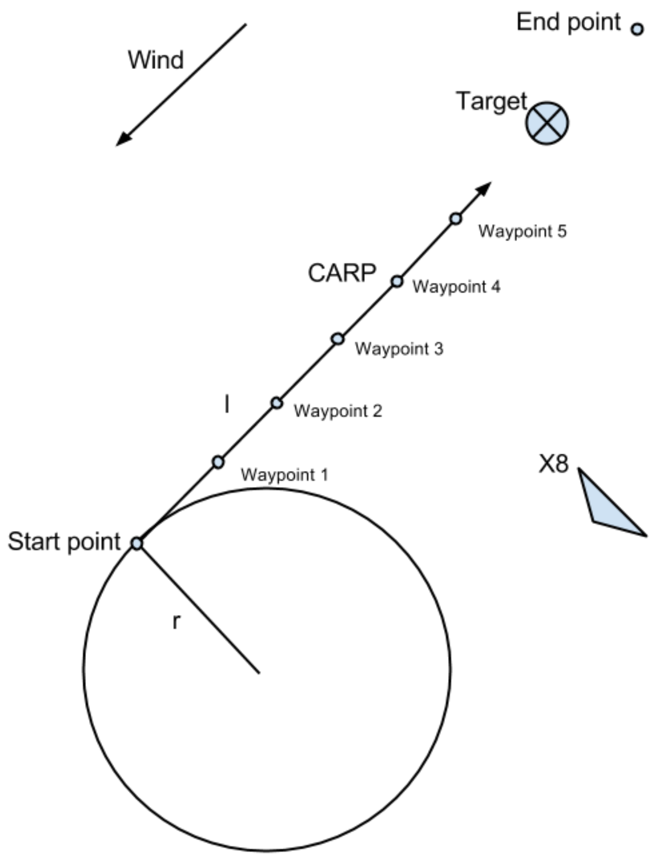


Figure 2.10: The Incremented-Straight-Line path

2.5.2 Incremented-Straight-Line

The Incremented-Straight-Line (ISL) path planner is based on a lot of the same theory as the SL approach, but uses several waypoints to draw up the straight line as is visualized in Figure 2.10. In this configuration, the waypoint controller will act more as a LOS controller and make some corrections to course as it increments the waypoints so that the path is followed more closely. The current waypoint is incremented when the UAV is close it.

2.5.3 Optimal

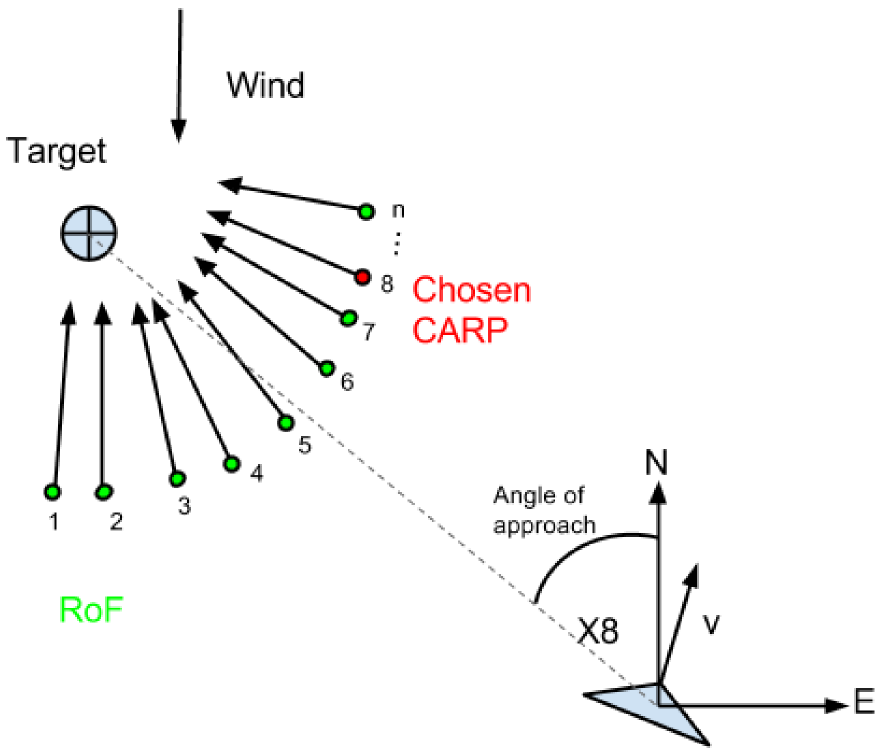


Figure 2.11: Example of choice of optimal CARP

The Optimal path planner differs from the line approach methods in Section 2.5.1 and Section 2.5.2 by not having a constant CARP. The payload computer chooses a CARP among the

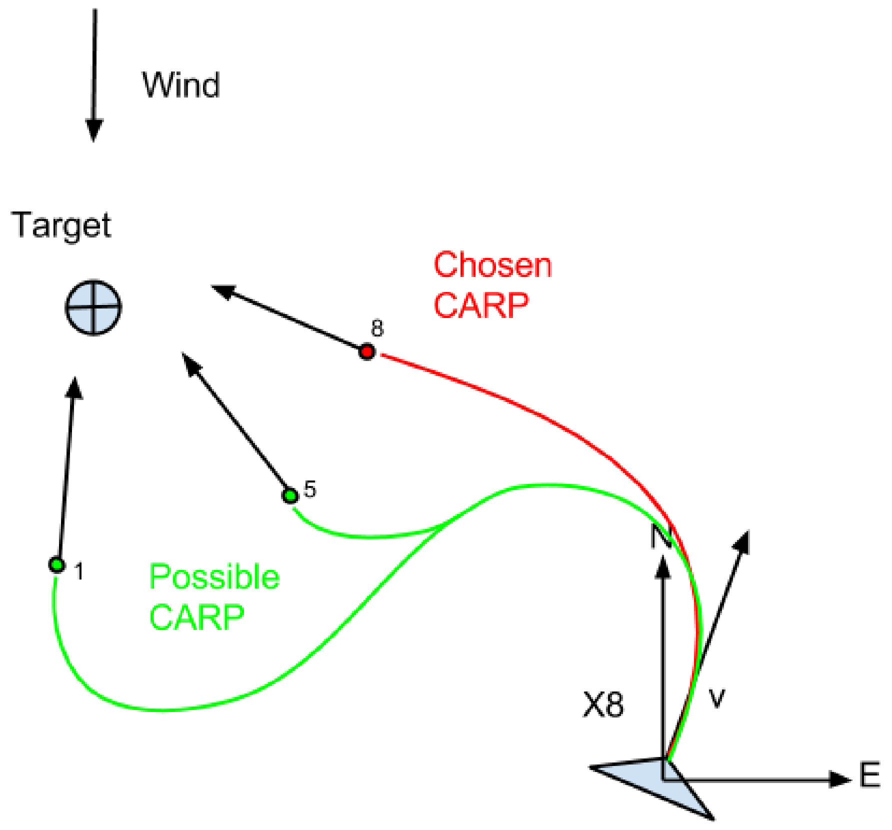


Figure 2.12: Some CARP paths are not as simple to follow

FARP at every few increments of the control algorithm. How often is given during initiation of the flight.

The FARP is the solution to the EoM for the trajectory described by Equation 2.10 given certain initial conditions (more in Section 2.1.3). As the analytical solution for this problem was not found, the FARP used in the optimization is a range of CARPs existing in the FARP. This range is called the range of FARP (RoF). The RoF is found through calculation of aerial-release points for n velocity vectors ϵ radians around the target based on the approach angle. ϵ is the angular difference between the first and last CARP velocity vector. The RoF is then n CARP instances of the FARP. This is visualized in Figure 2.11. Both n and ϵ are given on initialisation.

The optimization algorithm chooses the CARP simplest to reach by weighing each CARP in the RoF based on two criterion: proximity and difference in desired contra current velocity. A

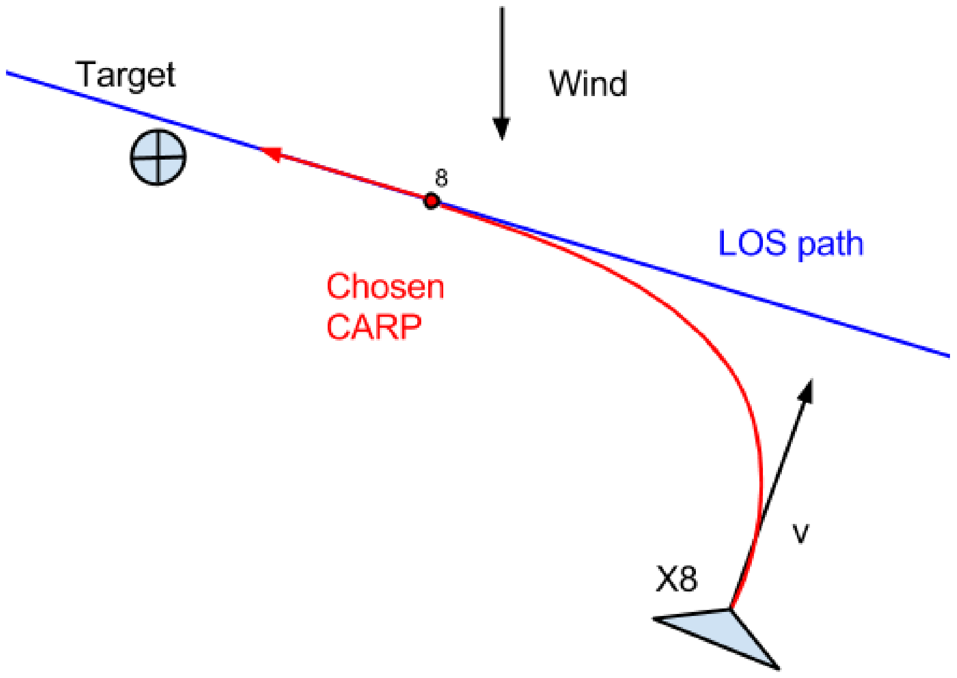


Figure 2.13: LOS path for optimal CARP

comparison of paths is seen in Figure 2.12. The UAV's velocity has to be rotated about the z-axis so that it is mirrored about the line made by the angle of approach. If it was not, the velocity weighing part of the optimization would always favour a CARP on the opposite side of the angle of approach, for instance point 1 in Figure 2.11, which is clearly not the simplest to reach.

Choosing new paths like this means that if the UAV is suddenly pushed off the path, it would not be bound to the currently chosen CARP, it would just choose a new CARP simpler to reach.

When the CARP has been chosen, a path is made for the LOS controller to follow, see Figure 2.13. The path is a straight line across the CARP in the same direction as the CARP velocity. The path contains an end point and uses the CARP as the start point, though the path extends beyond these.

There are several advantages with the Optimal path planner:

- It does not rely on a constant wind. It uses the newest wind estimate at all times.
- The UAV does not have to go to a start point first. It travels directly to the end point (via the CARP). It is also possible to go to a start point first, similar to the straight line path, to

be less affected by side slip.

- As the height might vary, this method takes height into account at every iteration.

There are also a few disadvantages

- The optimization might use a lot of computational resources, and might cause deadlines to be missed.
- The almost continuous updating of the LOS's path will in some cases make the UAV course oscillate.

2.5.4 Optimize-When-Success-is-Improbable

The Optimize-When-Success-is-Improbable (OWSI) path planner is designed to fend off the disadvantages of the Optimal path planner. As the Optimal path planner is based on optimizing very often, the OWSI will only optimize when it is improbable that the CARP will be reached or that the wind has changed sufficiently to change the beacon trajectory considerably. This will lead to fewer optimizations, further leading to less CPU usage and less path oscillations.

Whether to optimize or not is based on the UAV's current state, the CARP and the wind.

- If the wind has changed a considerable amount (amount given during initialization), the projected trajectory will be false.
- If the actual height is different to the CARP height, the target will be missed. A new CARP must be considered in the current height of the UAV.
- If the path error is large, the UAV will probably miss the CARP, either as the position is not reached, or as the velocity will be in the wrong direction.
- If the speed is not at its set value, the calculated trajectory is false.

The final three elements have to be considered with distance to the CARP in mind as the LOS, speed and height controllers need time to get to their set points. The following considerations are suggested to decide whether to perform a new optimization or not.

- If $w_{carp} - w > c_w$, where w_{carp} is the wind used to calculate the CARP, w is the newest wind estimate, and c_w is a constant.

- If $e_p/d > c_p$, where e_p is LOS path error, d is distance to CARP, and c_p is a constant.
- If $e_h/d > c_h$, where d is distance to CARP, e_h is the height error, and c_h is a constant.
- If $e_s/d > c_s$, where d is distance to CARP, e_s is the speed error, and c_s is a constant.
- If $(\mathbf{v}_d^b - \mathbf{v}^b)/d > \mathbf{c}_v$, where $\mathbf{v}_d^b$ is the desired velocity, $\mathbf{v}^b$ is actual velocity, d is distance to CARP and $\mathbf{c}_v$ is constant.

2.6 CARP Detection

It is important to accurately decide when the CARP has been reached in order to drop the beacon at the correct time. This is in general done in two ways, either by detecting when a perpendicular plane at the desired position has been crossed, or by making a sphere around the desired position and deciding when it has been breached (Beard and McLain, 2012).

In this thesis, a mixed version is used. The decision is based on the distance to the CARP. As we have a quite accurate and frequent position estimates (as evident from Section 7.4.3),the detection may provide an accuracy of 0.85 m when the UAV's speed is 17 m/s. This is calculated by dividing the speed measurement by two times the update rate, $V/(2r_{est})$. This is only valid when flying straight above the CARP as seen in flight 1 in Figure 2.14.

The logic of the decision is that the release should take place either when the measured distance is less than a decision radius or when the UAV is further away than in the previous distance measurement.

The decision radius should be the UAV's speed divided by between one and two times the state estimate rate(v/r_{est} to $v/(2r_{est})$).

From Figure 2.14 one can see that as flight 1 and 2 are flown perfectly above the target, a drop will occur within the expected accuracy. When flying close to the decision radius perimeter as in flight 3 and 4, the accuracy is not so high.

Figure 2.15 maps where the CARP is considered reached when an estimated state is within

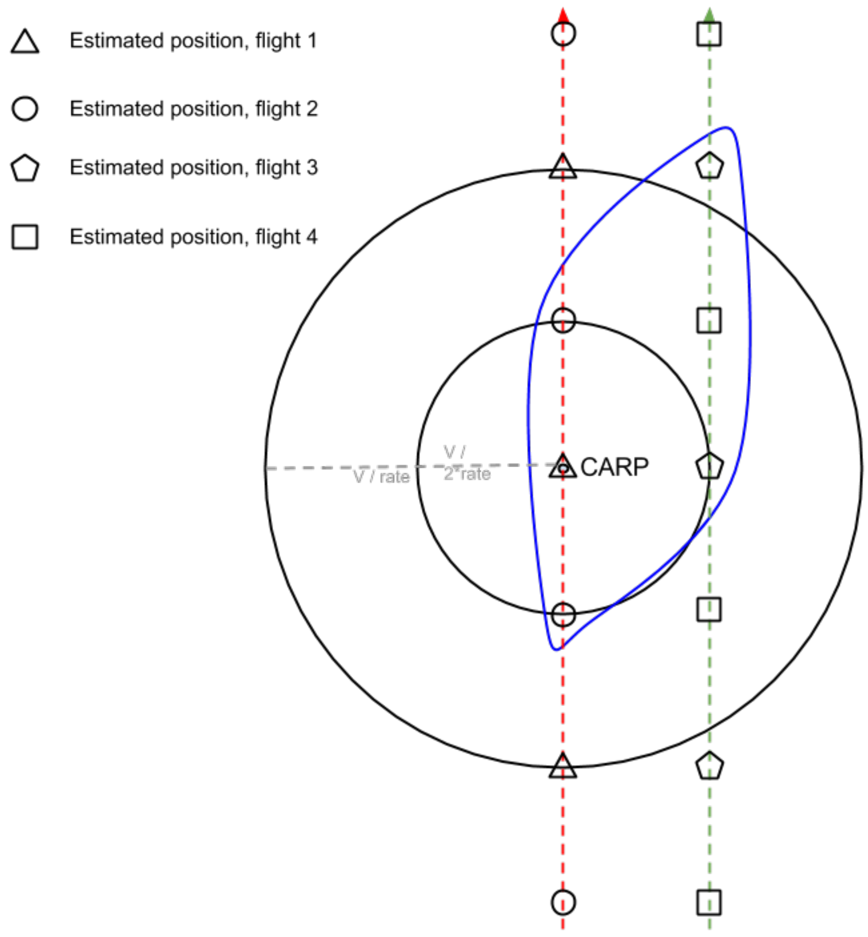

Estimated position, flight 1
Estimated position, flight 2
Estimated position, flight 3
Estimated position, flight 4
V / rate
V / 2*rate
CARP

Figure 2.14: Estimated positions for four flights close to target. The blue line shows at which estimated state the CARP is considered reached

the orange areas. The UAV has a velocity upwards in the picture but the UAV might not be positioned at the indicated start position. The red field shows how wide the decision radius may be.

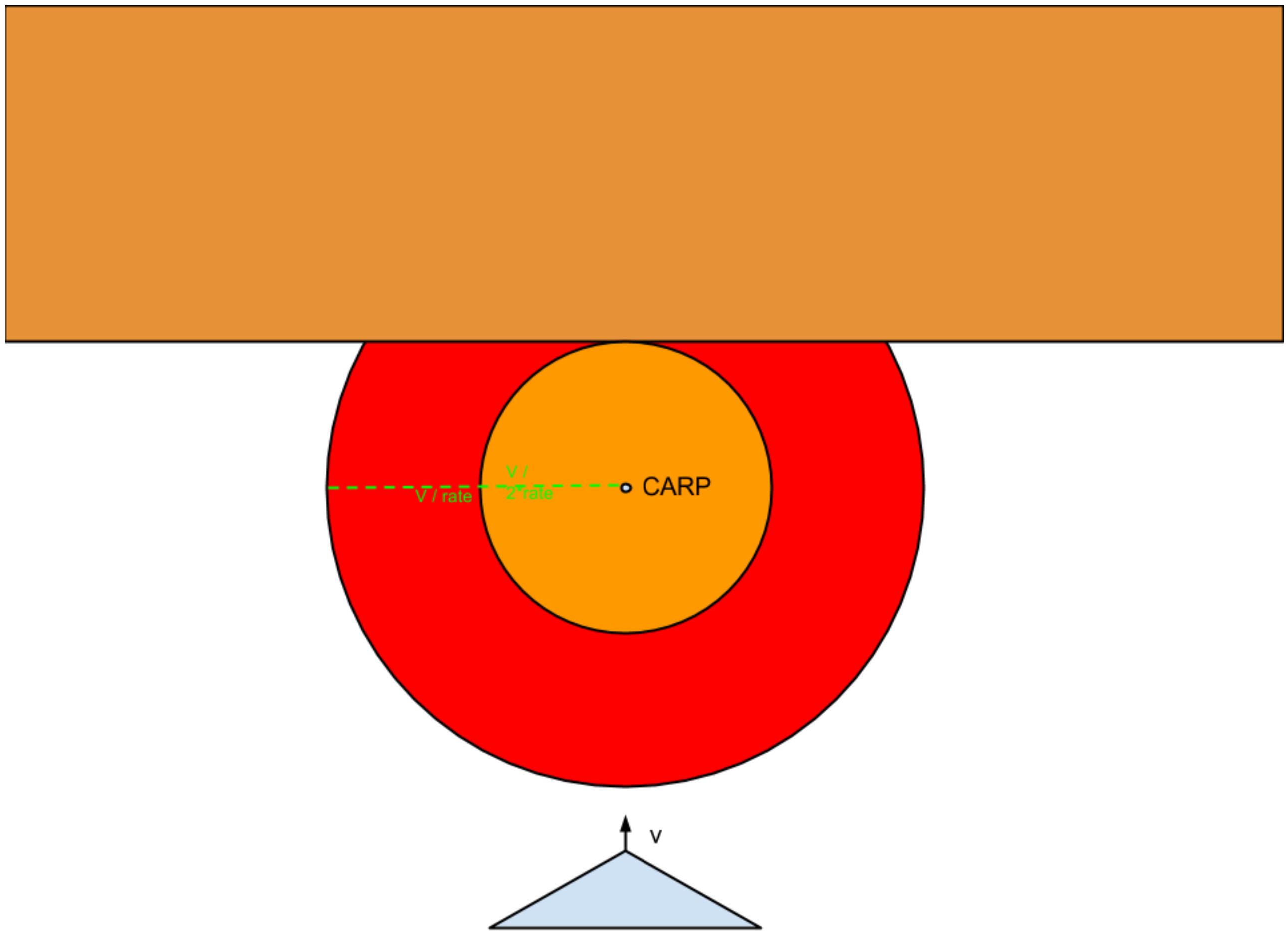


Figure 2.15: Map of release positions

Chapter 3

Hardware Implementation

This chapter will present the implementation of the hardware mounted on the X8, and the X8 itself. All off-the-shelf products are described in Section 3.1. To fasten the payload in the X8, some box or plate had to be designed. This is described in Section 3.2. Section 3.3 gives an overview of the PWM generator. This product was tailor-made during the thesis. The last section in this chapter, Section 3.4, gives an overview of the complete system by describing the hardware setup in three ways: communication, power supply and modularity.

3.1 Hardware Components

The X8 specifications and attributes are described in Section 3.1.1. The chosen payload computer was the BeagleBone Black. It is a well suited computer for this task, as it has enough computing capacity and does not use too much power. It is further discussed in Section 3.1.3, and information on its expansion board is found in Section 3.1.3. The mechanism that drops the beacon is described in Section 3.1.2. Information on the autopilot that was used can be found in Section 3.1.7, and the radio used in Section 3.1.4. In Table 3.1, all used hardware is listed.

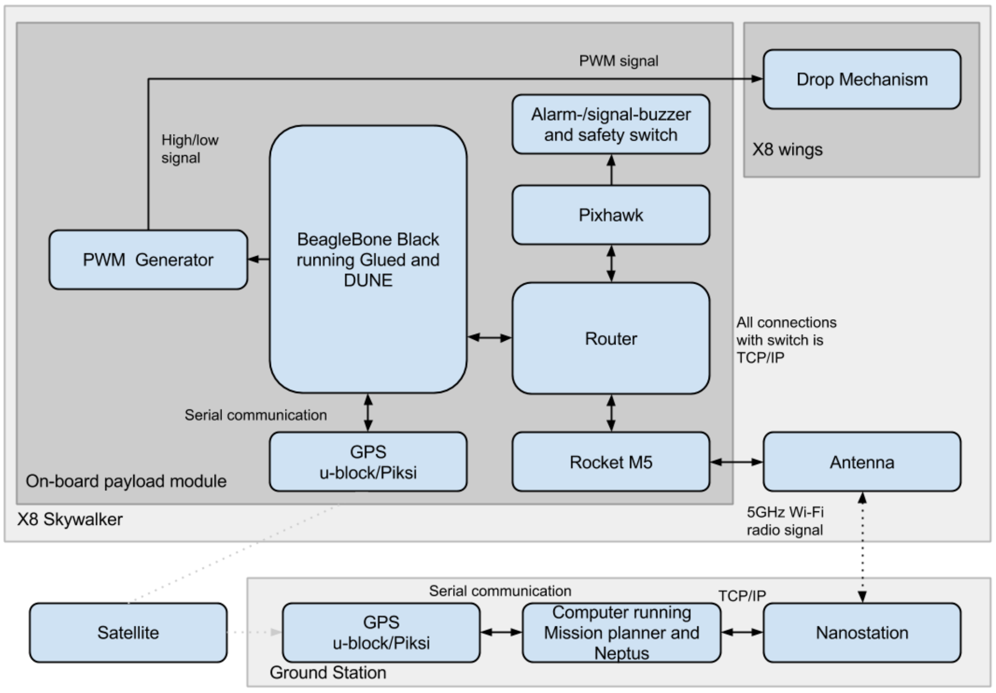


Figure 3.1: Hardware setup and signal flow

Part/module	Type/usage	Description
Skywalker X8	Vehicle	Fixed wing UAV platform
BeagleBone Black	Avionics computer	Computer as part of payload
Rocket M5	Radio	Communication with ground
Nanostation M	Ground station Radio	Communication with UAV
Pixhawk	Autopilot/Guidance	Used to follow waypoints
EFLA405	Drop mechanism	Used to drop beacon
On-board router	Router	Supporting LAN in UAV
PWM generator	Signal converter	Sending PWM signal to drop mechanism
Aluminium plate	Mounting structure	For mounting of payload

Table 3.1: Hardware parts list

3.1.1 Skywalker X8

Wingspan: 2120 mm

Weight: 880 g

Motor: 2820 KV730

Speed regulator: 60 A

Battery pack: 4S8000 mAh

Propeller: 12X6 13X8

Maximum take-off weight, according to third party: 3200 g

Maximum take-off weight, according to pilot: 4500 g

These measures are based on a building instruction manual made by a third party.

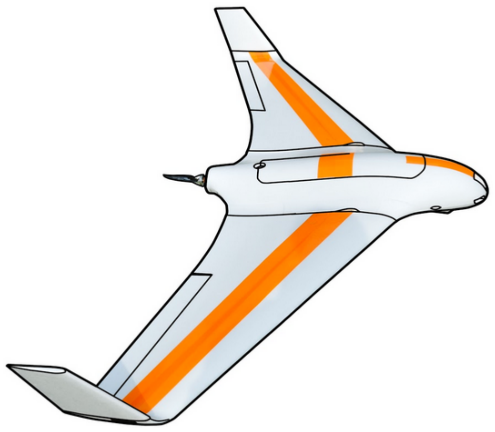


Figure 3.2: Sketch of Skywalker X8

The X8 is a small and inexpensive aircraft, which is one of the main reasons for choosing this platform. Modification of the aircraft was simple, as it is made from styrofoam. The shape of its hull was also advantageous. The glider shaped design gave flight advantages, requiring less power and the ability of keeping low speeds. The X8 also has a fairly large payload capability, compared to its size. The X8 is drawn in Figure 3.2. Its optimal airspeed is about 17 m/s, and uses a catapult-like launch device to take off, pictured in Figure 3.3.

For these reasons, the Skywalker X8 was a well chosen aircraft to use in this thesis. Modifications were made to the hull, enabling the X8 to drop a beacon from underneath its wings and its belly by attaching drop mechanisms there.

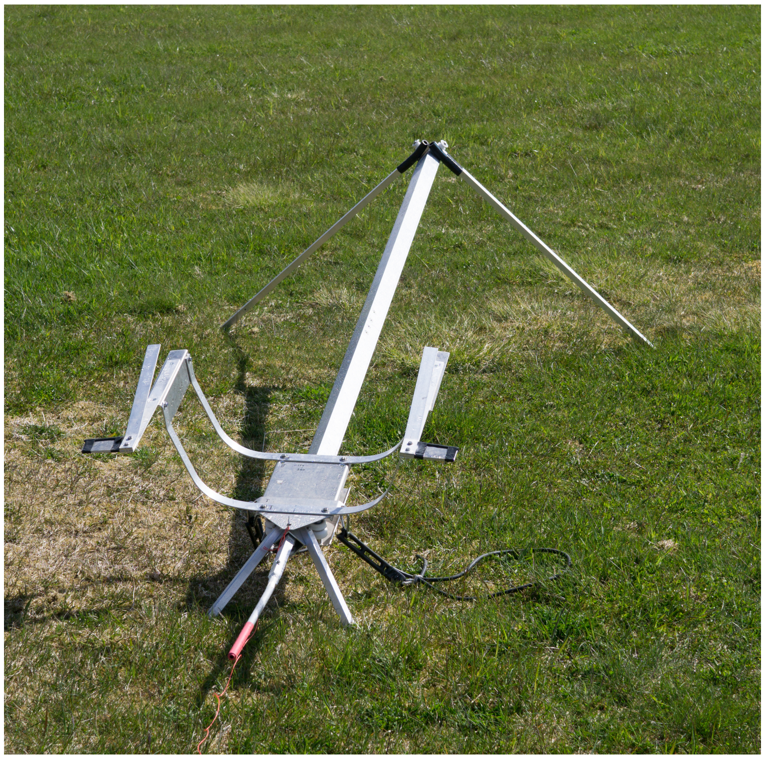


Figure 3.3: X8 catapult

3.1.2 EFLA405 Servoless Payload Release

To release the beacon, a drop mechanism was needed. It had to be light, quick, and not use much power. The EFLA405 Servoless Payload Release (henceforth called the drop mechanism) was chosen as it only weighs 18 grams and uses about 350 mA with 4.8-8.5 V (Servoless Payload Release, 2014).

The release of the payload was measured to take 0.6 s.

Placement of Drop Mechanism

To decide where to place the drop mechanism, several problems needed to be taken into account. First, the X8 has no wheels or undercarriage. It lands on the belly of the plane. The wings will also probably touch ground during landing. Second, the propeller's integrity is important.

Failure of the propeller is a major or catastrophic failure. Third, the drop mechanism could not interfere with the launch ramp. Fourth, the payload should not change the centre of gravity.

These considerations meant that the placement of the beacon and drop mechanism was important. Two placements were considered, either at the wings or at the belly of the plane. As placing the drop mechanism on the wings would interfere with the launch mechanism, the final placement ended up being on the belly of the plane. This meant that the engine must be turned off before releasing the beacon.

3.1.3 BeagleBone Black

The BeagleBone Black (BBB) was chosen as the payload computer. It is a small computer with most types of I/O available. The work in this thesis required serial communication protocols and Ethernet, which is available on the BBB.

There are many ways to power the BBB. The chosen design uses a voltage regulator on the cape (see the below section) to supply power through the P9 expansion slot pin one and three. Other available methods would have been to supply it through USB, Power Over Ethernet (POE) or DC Jack (cupped 5.4 mm) (BeagleBone Black Datasheet, 2014).

Hardware

Processor: AM335x 1GHz ARM® Cortex-A8

Memory: 512MB DDR3 RAM

Storage 4GB 8-bit eMMC flash

Graphics: 3D graphics accelerator

SIMD: NEON floating-point accelerator

Microcontrollers: 2x PRU 32-bit microcontrollers

Connectivity

- USB client for power and communications
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers

(BeagleBone Black Datasheet, 2014)

Cape

The cape is an expansion board for the BBB. It is shaped like the BBB, with a cut-in for the Ethernet port so as to fit on top of the BBB. The cape uses the P8 and P9 expansion slots on the BBB.

The cape is used as power supply and as an interface for I/O serial ports. It is powered by a battery, with a voltage capacity between 6.5 V and 18 V. A voltage regulator, R-78B5.0-1.5L, lowers the input voltage to 5 V (R78Bxx15L Datasheet, 2014). Three serial I/O ports are mounted on the cape. One for the Pixhawk (marked PX), one for GPS (marked GPS) and one optional port (marked OPT). There is also a port power supply for the Power Over Ethernet cable (see 3.1.8). All connectors are ATX 4 pins.

The printed circuit board (PCB) for the cape was designed by Kristian Klausen at ITK, and machined and assembled by the author in the cybernetics workshop. The PCB design and circuit diagram can be found in respectively Appendix K and Appendix J.

AM335x 1GHz ARM® Cortex-A8

The BBB's processor. It is made by Texas Instruments, and their description of it is available in Appendix F.

3.1.4 Rocket M5

A radio was needed to communicate with the ground station. The Rocket M5 was chosen. It uses Wi-Fi and supports TCP/IP. This was necessary as the IMC protocol requires TCP/IP. For more on IMC, see Section 4.1.3. The Rocket M5 has two 5 GHz radios and a throughput of 150+ Mbps (Rocket M5, 2014). As it has two radios, two appropriate antennas are needed.

3.1.5 Nanostation M

The UAV ground station needed a radio as well. It had to be compatible with the 5 GHz transmission from the Rocket M5 and support TCP/IP, so model NSM5 was chosen (Nanostation M, 2014). The Nanostation M was used only as a ground station radio at the lab as a Rocket M5 rig was set up on the airport used for testing.

3.1.6 On-Board Router

A router was placed in the UAV. Internal distribution of information via the IMC bus is then made possible. The router was made by Torkel Hansen and Artur Piotr Zolich at ITK and is run by an Atheros AR9331. It uses the Linux distribution called OpenWRT, which is good for networking on embedded devices (OpenWRT linux distribution, 2014). It is connected to the Pixhawk using a Fiber-Distributed-Data-Interface (FDDI) cabel which has a virtual-serial interface, so that all information to and from the Pixhawk complies with the TCP/IP standard.

3.1.7 Pixhawk

Pixhawk is an advanced autopilot system designed by the PX4 open-hardware project and manufactured by 3D Robotics. It features an advanced processor, a failsafe processor and sensor technology from ST Microelectronics which is basically an IMU, and a NuttX real-time operating system, which is delivering good performance, flexibility, and reliability for controlling the

X8. It is also small and light, making it well suited for small UAVs. Pixhawk offers its own complete flight control stack, however, it can be modified if needed (Pixhawk Autopilot, 2014).

It has several ways of communicating:

- 5x UARTs
- 2x CAN
- I2C
- SPI
- ADC
- Spektrum DSM / DSM2 / DSM-X
- Futaba S.BUS
- PPM sum signal
- USB

(Pixhawk Autopilot, 2014)

To control the servos, Pixhawk has 14 PWM outputs (8 with fail safe and manual override, 6 auxiliary, high-power compatible). For information on available software, see Section 4.2.2.

The Pixhawk uses a differential GPS with a 5 Hz update rate (GPS Modules, 2015). As it is differential, it's accuracy is 2.5 m in position (LEA-6 series u-blox, 2015).

3.1.8 Cables and Connectors

Due to different connectors on-board the UAV, cables had to be made.

The Rocket M5 requires POE, but the BBB's Ethernet port does not supply it. The BBB is normally not connected directly to the Rocket M5 except during early testing, so the need was not dire. The solution was to split the Ethernet cable between signal and power, by plugging the power part of it into the BBB's cape and the rest into the BBB's main Ethernet port.

A cable was made to connect the PWM generator into the BBB's cape. It has one DF13 connector in one end and a flat four pin ATX connector in the other.

3.1.9 Test Beacon

The test beacon was made so that drop accuracy flight tests could be performed. The test beacon had to be stable in free fall, meaning its centre of pressure must be behind its centre of gravity. The drag coefficient, mass and area had to be know.

A cone shaped test beacon was made from plastic with a part of the back milled out. It has a 20.25 degree angle from the centre line to the side. This led to a stable shape weighing 104 g with a drag coefficient of about 0.39 (Drag, 2015). The front facing area is 0.00636 m. These numbers led to a b of 0.00152 using an air density of 1.225 kg/m^3. Its stability was confirmed by throwing it off the roof of a building. The test beacon is pictured in Figure 3.4 and 3.5.

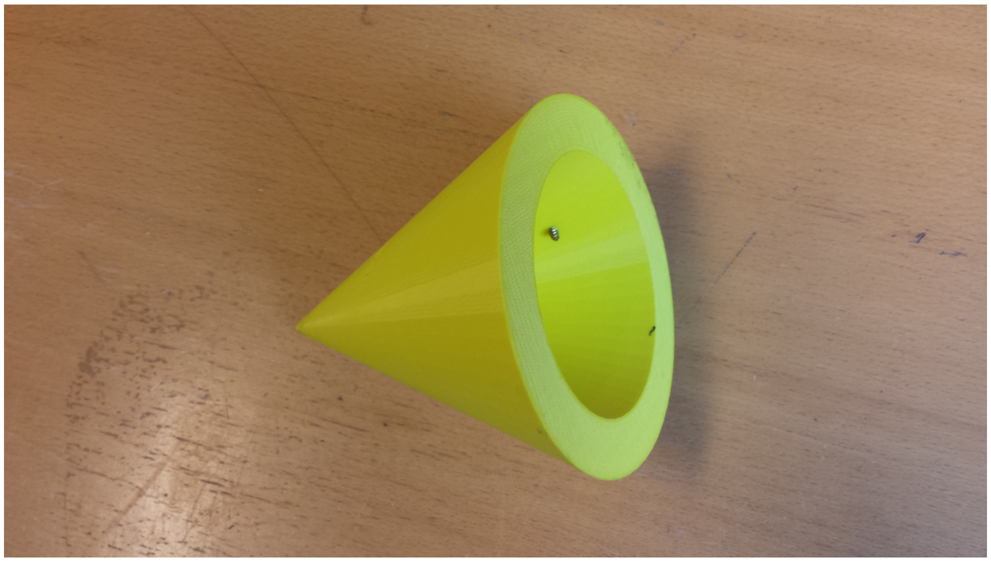


Figure 3.4: Test beacon with the milled-out back visible

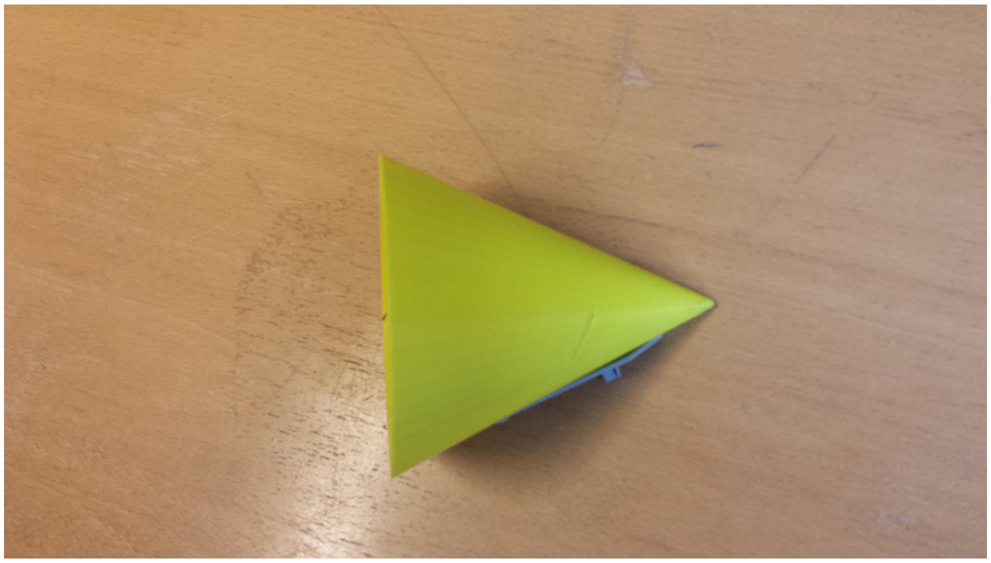


Figure 3.5: Test beacon

3.2 Placing the Payload Components

The hardware on the X8 had to be mounted in such a manner that it would not change the dynamics of the X8 too much. Weight and placement of the payload had to be considered carefully as the X8 has maximum payload capacity of 4.5 Kg. Mounting and unmounting had to be simple. Velcro was therefore used on all components to secure them in the payload bay. Components that were not suited to be directly velcroed in the UAV were raised and mounted to a thin aluminium plate, which was then velcroed to the UAV. In the final design, only the BeagleBone Black was mounted on one such plate.

3.3 PWM Generator

A PWM input is required by the drop mechanism. This was achieved by making a small PWM generator using a microcontroller (MCU) to supply the PWM signal. The final design can be found in Appendix K and Appendix J.

To design the circuit, Altium Designer and Eagle was used. Programming of the PWM generator is described in Section 4.3.

The ATtiny85-10PU MCU was chosen. This is a cheap, simple, available MCU that does not consume much power.

3.3.1 Programming the ATtiny85 MCU

The ATtiny85 was programmed in Atmel Studio 6.2 using the test board AVRATSTK500 and a JTAG ICE MK. II until a JTAG compatible port was included in the PCB design, see Figure 3.6 and 3.7

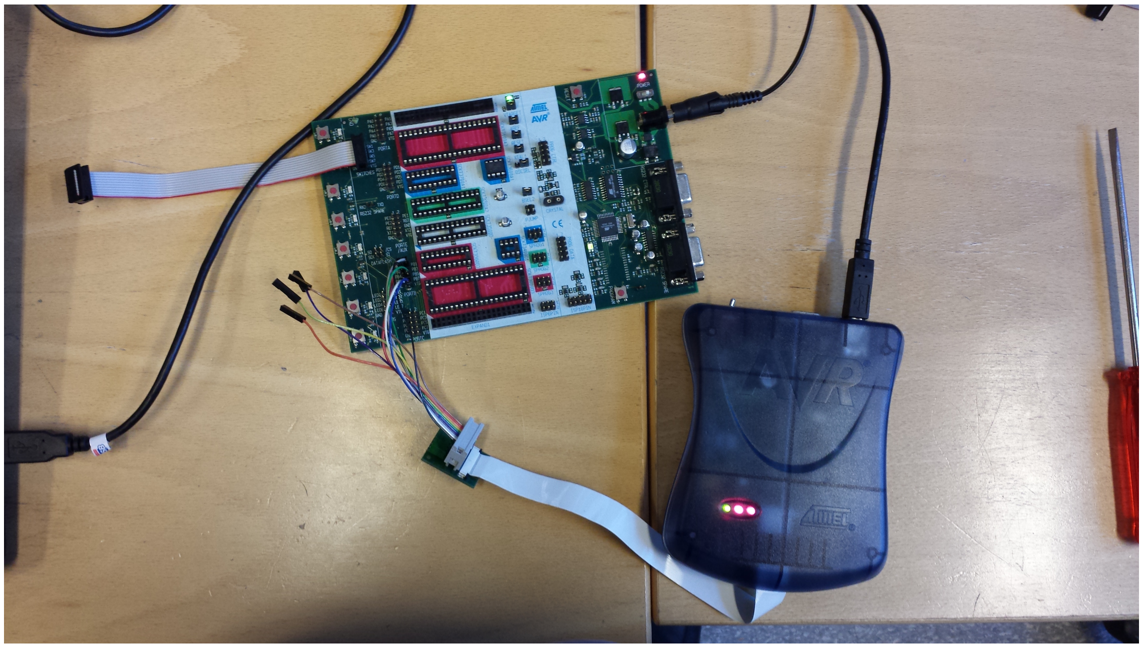


Figure 3.6: Programming the ATtiny85 using the AVRATSTK500 and JTAG ICE MK. II

3.3.2 First Design

The size of the PCB was approximated to 40 x 40 mm, but the final measures of the first design was 30 x 30 mm. Through-hole PCB was chosen as the existing equipment used to program the MCU is designed for through hole components. In the final design, the JTAG port was implemented directly on the PCB, and the design could then have been changed to use a surface-mounted MCU.

The cybernetics workshop uses a a milling machine to make PCBs, and the available PCB plates were two layered. The first design had one four-pin header for both input and output. This was not a good design, as the cables did not match the designed template. Four holes were drilled for screws, but this was not strictly necessary to attach the PWM generator in the payload compartment. The lack of a port for programming meant that there would be no software updates after soldering. This might be crucial, as the timing and integrity of the release mechanism would be a source of error for the point of impact of the beacon.

3.3.3 Final Design

On the second design, the size was changed to 35x28mm. The input/output header was split into two separate headers. The output was chosen to be a three-pin header, as it would fit the drop mechanisms connector. A DF13 five pin connector header was used as input. This is the same header that the Pixhawk uses. This was chosen because of its small footprint and strong fit, although detaching the cable may rip the connector off the PCB.

Only one hole was made for attaching the board to the UAV on the second design. This led to a smaller PCB design compared to using four holes. A port was added for the JTAG which made it possible to update the MCU software. This also retired the AVRATSTK500 test board, as we now could program the ATtiny85 directly on the PCB.

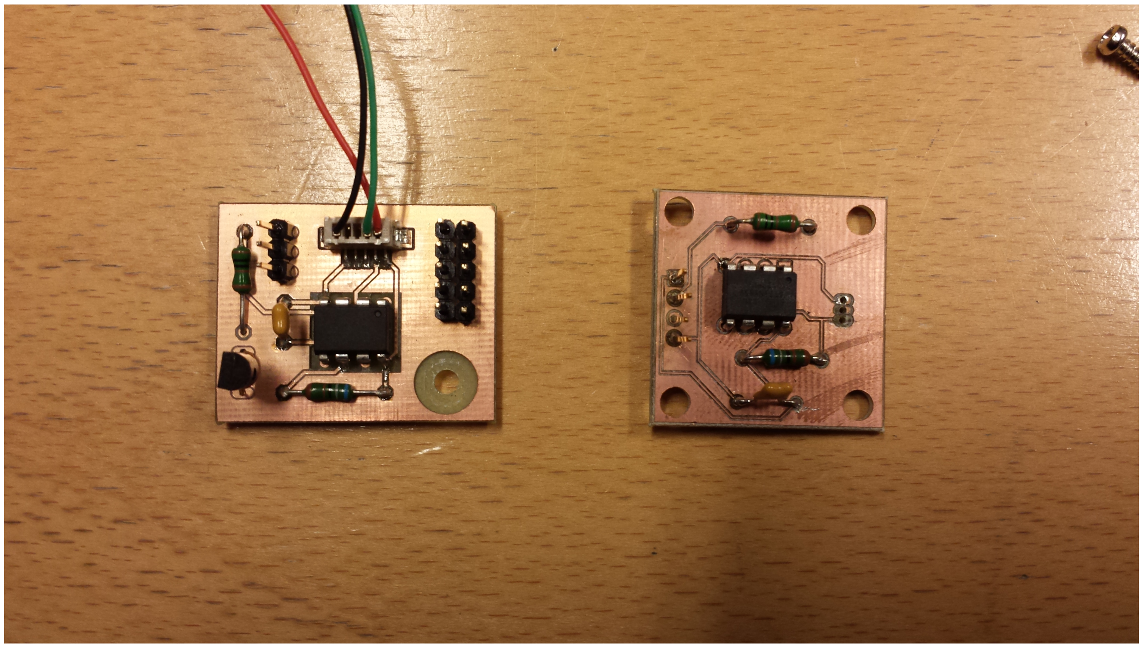


Figure 3.7: Final design on the left, first design on the right

3.3.4 Testing

The input signal was at first produced manually. The GPIO input was, via a wire, connected to either GND or 5 V to test the PWM generator's reaction. A test program was later made in DUNE to see how the PWM generator and BBB cooperated. This is described in Section 6.1.2. An oscilloscope was used to verify the PWM-signal.

3.4 Hardware Overview and Setup

This section will try to give an overview of the entire hardware setup. In Section 3.4.1, the connections and communication are explained, then the modules will be described in Section 3.4.2. The last section, Section 3.4.3, shows how all components are powered.

3.4.1 Connections and Communication

The IMC is the main communication protocol in this project. It works over the TCP/IP protocol, and should therefore be connected in a compliant way. The BBB and the Rocket M5 has RJ45 connectors, and connects directly to the on board router. An FDDI cable was fashioned for the Pixhawk autopilot so that it could communicate directly with the router as well, though if using Neptus and APM Mission Planner at once, the Pixhawk should be connected to the PX port on the BBB's cape. It is important to note that the ground station is also communicating via the IMC bus, and that contact between ground station and X8 is made using the radio link. This connects all the main components to the IMC bus except the PWM generator with drop mechanism.

The PWM generator has a simpler way of communicating, and gets a high/low signal from the general purpose I/O (GPIO) on the BBB. This signals the PWM generator to start the PWM signal, which in turn will make the drop mechanism release the beacon.

Figure 3.1 provides an overview of the signal flow in the system, while Figure 3.8 shows the physical system.

3.4.2 Modules

To better divide areas of responsibility among the components, several modules were made. By building the UAV hardware in a modular way, errors will be less likely to spread, and it may be possible to introduce new modules later or take some away. The hardware modules, and their responsibility, as used in this project, are described below.

Payload Computer

The payload computer's responsibility is to run the runtime environment, DUNE, and its tasks. Tasks will vary depending on the current mission. The release mechanism is connected to the payload computer. The main communication channel for the payload computer is the on board LAN, connecting it for example to the autopilot and ground station via the IMC bus.

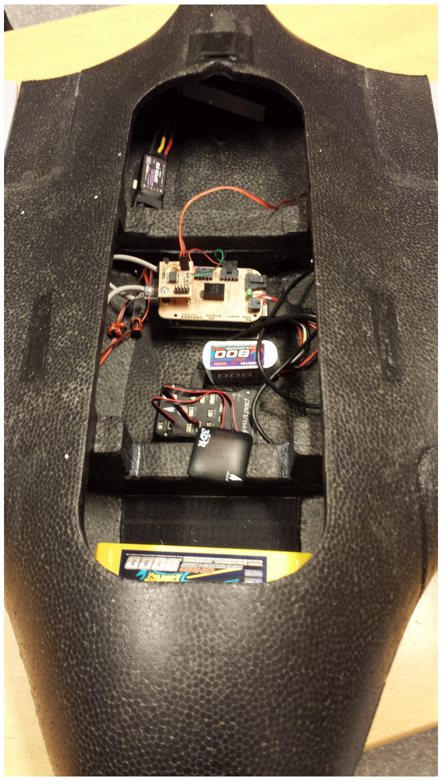


Figure 3.8: Payload

There will also be modularization inside the payload computer, achieved by running different tasks in DUNE. These tasks' responsibility will be, amongst others, to:

- Receive information from the Pixhawk
- Calculate the next waypoint
- Release the beacon
- Run flight controllers

Release Mechanism

Consisting of the PWM generator and drop mechanism, the release mechanisms responsibility is to drop the beacon when a signal is applied. Its only interface is to the payload computer via a high/low signal.

Autopilot

Navigation by waypoints received from the payload computer is the autopilots responsibility when the waypoint controller is used. If other controllers are used, the autopilot controls the roll angle or uses the bank-to-turn controller. It is also able to make wind and state estimates and making them available for the payload computer. Communication is made via TCP/IP or serial connection to the payload computer.

Radio

The radio keeps contact with the ground station. It has sufficient range to manage all planned missions and is fast enough to send all necessary data. It communicates via the on-board LAN and the radio Wi-Fi. A radio for the ground station is also required.

Ground Station

To keep ground personnel informed and able to send commands to the payload computer, a ground station is needed. It consists of a computer running necessary software and communicating via a radio. The software includes Neptus, mission planner and other software specific to the mission.

3.4.3 Power Supply

To power all components, a power supply was designed. As it was supposed to be installed in a UAV, it had to be based on either a battery or on a generator. There is no generator available in the X8 as it uses an electrically powered engine, so a battery-based power supply had to be designed.

Two different voltages was needed as the Rocket M5 uses 12 V and all other payload hardware uses 5 V. 12 V was delivered from the router vis POE.

For the 5 V regulator to be able to accommodate most variations of batteries, a wide input voltage was desired. The regulator chosen was R-78B5.0-1.5L from Recom, with an input voltage of 6.5-18 V and output voltage of 5 V. The maximum possible current draw is 1.5 A (R78Bxx15L Datasheet, 2014), which should be sufficient. This regulator works with all the batteries available at NTNU's UAV-lab.

The battery is wired to the BBB's cape where the voltage regulator is installed. 5 V is then input to the BBB using its expansion slot, P9. The power is also shared with the PWM generator via a cable from the cape. The PWM generator will then supply the drop mechanism via its servo cable.

The Pixhawk has its own 5 V voltage regulator, which is connected directly to the main engine battery pack. This is to ensure that the UAV is able to be controlled manually by remote control if needed.

Chapter 4

Software Implementation

The software used in this project can be divided into three groups, software on the X8, described in Section 4.1, software on the ground station, described in Section 4.2 and software on the PWM generator in Section 4.3. The PWM generator gets its own section because of its specific area of use. In Section 4.4, the DUNE tasks are described. The final section contains the software installation and setup. A system overview is provided in Figure 4.1.

4.1 Software for the X8

Software on the X8 was implemented in two hardware components, the BBB and the PWM generator. All other components were already programmed and needed only firmware updates to work as desired. The operative system (OS) and runtime environment will be described in Section 4.1.2 and in Section 4.1.3, respectively.

4.1.1 Software on the BeagleBone Black

The BBB is basically a fully functional computer, able to run Linux distributions (BeagleBone Black Datasheet, 2014). More specifically, Linux can be compiled to run on the BBB's ARM A8

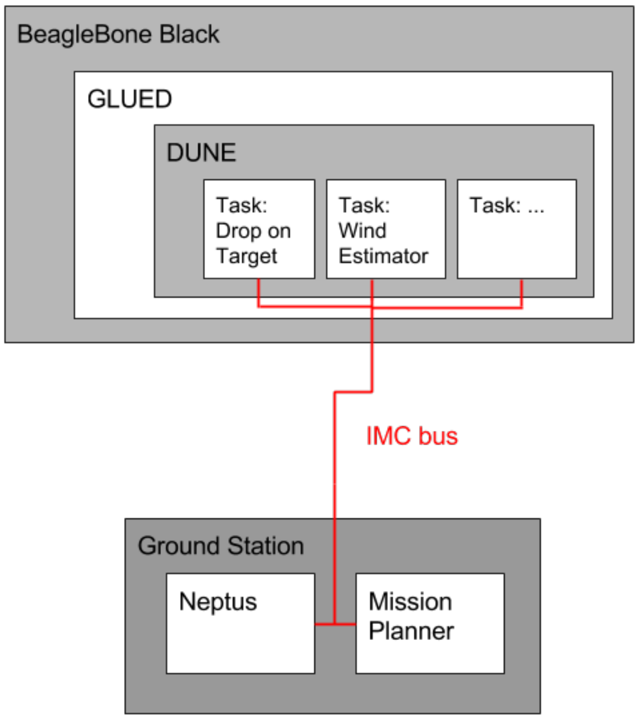


Figure 4.1: Overview of software and communication

CPU. BBB also comes with RAM, flash memory, BIOS and firmware to support basic functionality. GLUED was chosen as OS distribution. It was created at LSTS, designed mainly to be able to run DUNE, while still having a small footprint and low processor demands.

The BBB use DUNE as its runtime environment. DUNE uses C++ code to generate tasks and takes an .ini file to decide which tasks to run. Several such tasks were made during the thesis, and they are described in Section 4.4.

4.1.2 GLUED

GLUED is a Linux based OS. With its small footprint it is well suited for embedded systems. As GLUED is based on standard Linux, it is not real time. However, it was still responsive enough to run the DUNE tasks created during this thesis, and is designed to be close to real time. The only element in the UAV that has to be real time is the Pixhawk, and it has its own real-time OS, RTOS (Pixhawk Autopilot, 2014).

GLUED is compiled using a configuration file. This file specifies which platform to run on, IP address and other information about the computer. ARM Cortex A8 was chosen as this project's platform.

Installation is described in Section 4.5.1 and in Appendix D.

4.1.3 DUNE: Unified Navigational Environment

> *DUNE: Unified Navigational Environment is the runtime environment for vehicle on-board software. It is used to write generic embedded software at the heart of the vehicle, e.g. code or control, navigation, communication, sensor and actuator access, etc. It provides an operating-system and architecture independent platform abstraction layer, written in C++, enhancing portability among different CPU architectures and operating systems.*
>
> (Dias, 2014)

DUNE runs tasks. Each task is a process that runs in the runtime environment. The tasks have their own memory space and communicates by messages. LSTS has made their own messaging protocol called the Inter Module Communication protocol (IMC) (Section 4.1.3).

DUNE is designed for embedded systems. Running on multiple variations of independent operating systems. The footprint is also considered small, being less than 16 MB (Dias, 2014).

Among the many DUNE tasks available, some are especially noteworthy. The Ardupilot task manages communication with the Pixhawk and translates all Pixhawk data from MAVlink to IMC. The transports tasks manages UDP and TCP layer information, where, among other functionality, you can decide which entities can send certain messages. There are also several controllers available to enable and tune.

Installation and compilation is described in Section 4.5.1 Appendix E.

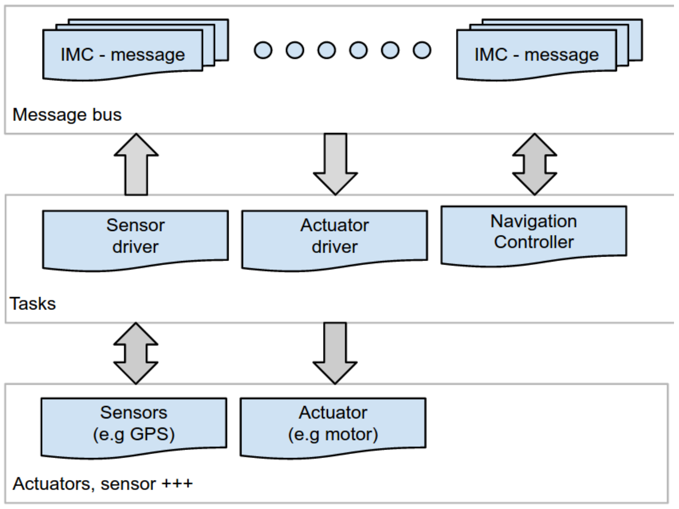


Figure 4.2: IMC task interaction principle

Inter-Module Communication Protocol

The bus based IMC protocol, which itself is based on TCP/IP, handles all communication between modules. These modules can consist of tasks on the same device, or tasks on another device entirely. This way, all modules on all vehicles, ground stations or other devices can communicate if they are connected to the same local area network (LAN). The IMC interaction principle can be seen in Figure 4.2.

The IMC protocol is message based. There is a big (and growing) database over different types of IMC messages. Each message consist of an ID and some data. Messages are also called packets. The packets consist of header, data and footer. Each of these include fields, which contain one variable, like a string or integer (IMC Specification, 2014).

4.1.4 MAVlink

MAVlink is the protocol used between the Pixhawk autopilot and the Ardupilot DUNE task. More information can be found at MAVLink Micro Air (2015).

4.2 Ground Station Software

Software used on the ground station will be discussed in this section. The main programmes used is the APM:Planner 2.0/Mission Planner, Section 4.2.2, and Neptus, Section 4.2.1.

4.2.1 Neptus

Neptus will run on the ground station. This is a cross-platform program for unmanned operations, with a lot of different applications and functions. The most general functions of Neptus is watching the IMC bus and sending IMC-messages (LSTS Toolchain, 2014).

Among other useful functionality is the ability to translate autopilot data sent over the IMC bus into a simulation environment where it is possible to oversee how the communicating vehicles behave, including map positioning, speed and attitude. It is very graphical as one can view the ongoing mission on a map, including a trail behind the UAV. It is also possible to give direct commands to any vehicle during the mission, like loiter or waypoint (LSTS Toolchain, 2014). In specific versions of Neptus used during this thesis, commands are found for placing a target and starting a drop mission.

4.2.2 APM Autopilot Suite

APM Autopilot suite are programs designed to be used with the Pixhawk (APM, 2014). The suite consists of firmware and software needed for working with the Pixhawk hardware. Different firmware versions can be chosen depending on platform. For fixed-wing UAVs, APM:Plane is the appropriate firmware. The software, APM:Planner 2.0 or Mission Planner, makes it easy to assign waypoints for the Pixhawk to follow. It is also possible to calibrate the autopilot and view live data or logs via the suite (APM Planner 2.0, 2014).

4.2.3 Ardupilot SITL

To run SIL simulations, Ardupilot SITL was used to simulate the Pixhawk autopilot. It works by acquiring the inputs normally sent to the Pixhawk autopilot module, and responds in a similar manner by sending the required signals to JSBsim, see Section 4.2.4. ArduPlane version 3.2.3 was used for testing in this thesis. For github directory and install instructions, see ArduPlane Autopilot (2015).

4.2.4 JSBsim

JSBsim (Berndt and Peden, 2015) is the physics simulation program used in SIL for this thesis. It receives autopilot inputs from Ardupilot SITL (Section 4.2.3) and returns flight data. In this way it allows Ardupilot SITL to work normally. The current publicly available version at 4.5.2015 was used (no version number is provided).

4.3 Programming of the PWM Generator

As the PWM generator is a simple device composed of general electrical components and an MCU, it was programmed directly using the proprietary software made by Atmel, Atmel Studio 6.2, avoiding the use of an OS. Atmel Studio 6.2 is a programming environment based on Microsoft Visual Studio.

The program running on the MCU is quite simple, as it consists of several busy wait cycles. Pseudo code of the program can be found in Algorithm 1, while the full code is available in Appendix H.

Algorithm 1 PWM generator main loop.

```
1: while True do                                          ▷ Always run
2:     if high_signal_from_BBB then
3:         RUN_PWM_ROUTINE()           ▷ Sends a PWM signal to the drop mechanism
4:     else
5:         set_PWM_high_cycle=0                     ▷ Make sure PWM signal is off
```

It is important to note that the PWM signal is active high.

The *run_PWM_routine()* function sends the PWM signal. This is done by setting a period and how much of the period should be a high signal. By incrementing the high signal size over time, the movement of a radio controlled lever is mimicked, making it a smoother transition for the drop mechanism.

The *while{True}* loop is polling for a high signal from the BBB (the test is in the next line). It does not include a wait or sleep function, so the program has the shortest possible period. This means that the PWM generator has the highest possible power consumption and the lowest possible response time.

A few possible improvements exists. The *set_PWM_high_cycle=0* is unnecessary as it is set to zero in the *run_PWM_routine()*. Another improvement would be to use normal sleep functions in the *run_PWM_routine()* instead of busy work. This would improve readability and possibly power consumption, depending on how the normal sleep function is implemented in Atmels libraries. A third possible improvement would be to make the *while{True}* periodic by sleeping until a certain time, for instance *Now()+100* ms after it has executed the while loop contents. This would make the program more schedulable and predictable. It would also make the PWM generator consume less power. The downside of this third improvement is higher response time.

4.4 DUNE tasks

Several DUNE tasks were needed in this thesis to produce the expected accuracy and functionality. They are described in this section. The .ini file mentioned in the tasks should be called by the UAV's system name (found while performing point two in Appendix D) to make it run on BBB startup. In this thesis the system name was ntnu-x8-006.ini.

4.4.1 The Actuator-Output Dune Task

Also known as *Actuators/LogicOutput/BBB*.

This task was created to switch an output on the BBB. Its inputs (members of the *m_args* struct) are

- *int pin:* Select which pin to control
- *int init:* Initial output on pin
- *int name:* Name of PowerChannel

This task sets initial values and initializes the output of the specified GPIO pin in *onResourceAcquisition()*, which is run on resource acquisition when starting the task.

The only consume function is run when receiving a *PowerChannelControl* message. The message contains the required output, and the output is promptly set. An answering *PowerChannelState* message is then sent to confirm the status of the GPIO channel.

4.4.2 The Wind-Estimator Dune Task

Also known as *Navigation/WindEstimator*.

The Wind Estimator task is based on the wind estimator described in Section 2.3. Its initialisation inputs (found in *m_args*) are

- *int sample_window_size:* Size of the sample window used to find the normalized observability gramian
- *double trustedlim:* The size of the observability gramian when the measurement is trusted (ϵ in Johansen et al. (2015), found to be 0.001 in simulations (Section 5.1.2))
- *double q_multi:* a multiplier for the Q weight matrix used for adjusting the size of Q in the Kalman filter.

To communicate with the other tasks, the wind estimator uses two IMC messages: *EstimatedState* and *EstimatedStreamVelocity*. The *EstimatedState* message contains information on the current estimated state of the UAV and is output from the autopilot task, while the *EstimatedStreamVelocity* contains wind information. The *EstimatedStreamVelocity* is the wind estimator's main output, and two different entities of this message are used. *m_wind_estimated* and *m_wind_at_the_moment*. The first entity is the Kalman filtered wind estimation and the latter is the raw wind measurement.

Upon receiving an *EstimatedState* message, it is saved in *m_estate*, and the *handlIas()* function is run. This function contains the main elements of the wind estimation theory in Section 2.3. It calculates the time since previous run (*m_dt*), updates the **C** matrix (*m_C*), updates the Kalman filter (*m_P, m_S, m_K*), calculates the estimation error (*m_e*), estimates the wind (*m_w*) and increases the observability gramian (*m_G*). Finally, the task transmits the estimated and current wind, then determines if the measurement is trusted based on the *trustedLim* and observability gramian.

Future functionality that this task could have would be to send out the calculated angle of attack and side slip angle. The calculations are available, but no message is sent for the results.

4.4.3 The Drop-on-Target Dune Task

Also known as *Autonomy/DropOnTarget*.

The main task running on the UAV is the drop on target task. It is the implementation of the approach methods based on theory in Section 2.3, 2.4 and 2.5. The CARP is also calculated as is described in Section 2.1. The task's inputs (found in *m_args*) are

- *double accepted_distance_to_start_point:* Acceptance circle around long stretch start point
- *fp64_t start_point_distance_from_carp:* Distance from CARP to end point
- *fp64_t end_point_distance_from_carp:* Distance from CARP to start point
- *fp64_t radius:* Radius of the loiter
- *double speed:* Speed set point during mission
- *fp32_t max_current:* Maximum motor current allowed for drop
- *fp32_t glide_time:* Glide time needed to stop the motor
- *fp32_t drop_time:* Time from drop signal send until dropped
- *fp64_t drop_error:* Drop within circle of this radius
- *fp64_t percent_accurate:* Percent accurate turning direction towards CARP

- *fp32_t release_height:* Drop release height
- *uint16_t altitude_accuracy:* The accuracy required from the height controller
- *uint16_t connection_timeout:* Response time before failure of *followReference*
- *uint16_t increments_input:* Increments for incremented LS guidance
- *std::string guidance_mode_input:* Guidance mode for mission
- *uint16_t safe_height:* High enough to be safe
- *bool bank:* Controller mode for mission (FBW or built in waypoint)
- *int counter_max:* Max counter for simulation
- *double dt:* Step size for CARP calculation
- *bool use_wind_est:* True to use wind estimator, false to use autopilot wind estimator
- *double opt_circle:* How many radians to use during optimization
- *int opt_points:* How many points to check during optimization
- *bool direct_to_opt:* True to go directly to target
- *bool optimize_once:* Optimize once (*true*) or continuously (*false*)
- *int opt_rate_inverse:* Inverse of the optimization rate
- *double w_pos:* Weighing of position during optimization
- *double w_vel:* Weighing of velocity during optimization

The task is set up as a state machine, as can be seen in Figure 4.3. It starts in the upper left corner, and the only interface input is either to send a target or to cancel the plan. To cancel a plan, a stop signal must be sent from Neptus on the ground station. If a plan is cancelled, the state machine goes back to the start state. If a new target is received during any part of a mission, the state machine goes into the second state, *Run function Consume(Target)*, making the UAV in practice restart and try to hit the new target.

State 1: Waiting for *Target* Message

This is a passive state wherein the UAV is loitering while waiting for a *Target* message. If the UAV is in *Guided* mode, the transmission to *FBWB* (fly-by-wire-b) mode might fail, so it is important to be in *Loiter* mode. The message is sent from Neptus via the *drop now* command found when

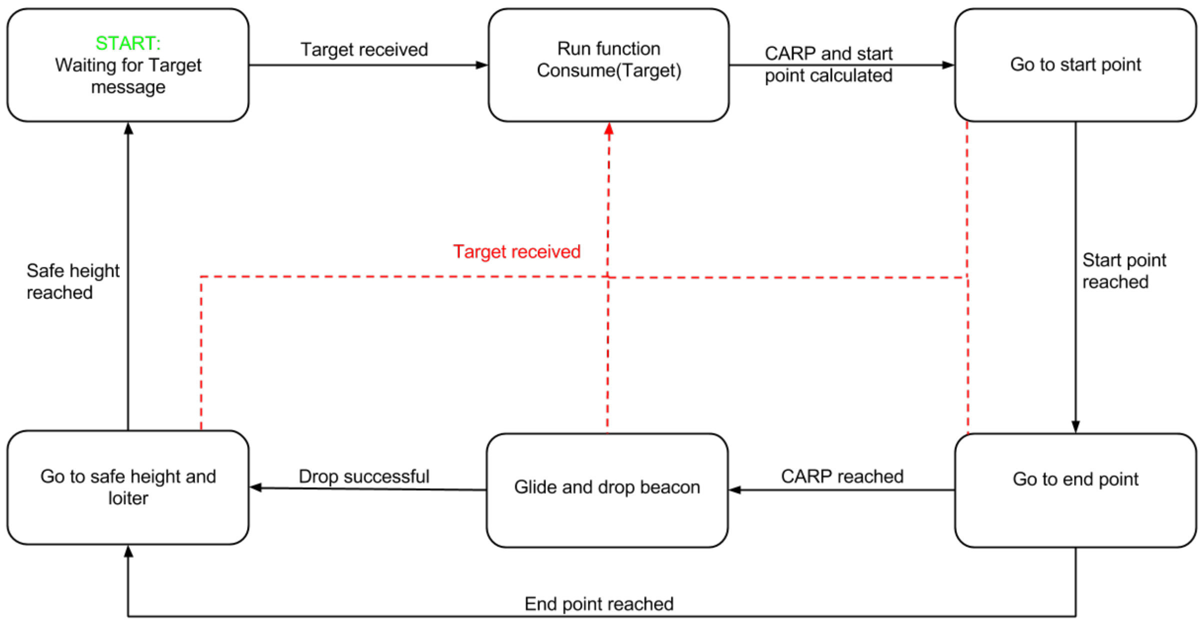


Figure 4.3: The drop on target state machine

using the drop overlay on the *ntnu-fixed-wing* console. That the state is passive means that the UAV can perform whatever command it is sent, like travelling a route to search for sea ice. Having the start state set up like this leads to good modularity.

State 2: Run Function *Consume(Target)*

The *Consume(Target)* function is used to start the state machine after waiting in State 1. The function calculates the CARP, start point, end point and first way-point. These are all the calculations needed for a drop using the LS path planner.

State 3: Go to Start Point

The next objective is to go to the start point. This state consist of going to a given waypoint and loiter around it. The waypoint has been calculated in the *Consume(Target)* function, and is placed *start_point_distance_from_carp* away from the CARP, and *radius* to the side, such that the UAV will get to the start point of the long stretch with the correct velocity, see Figure 2.9.

While travelling to the waypoint, the state machine waits for four statements to be true

- The latest *FollowRefState.proximity* has its *PROX_XY_NEAR* flag set
- The latest *FollowRefState.proximity* has its *PROX_Z_NEAR* flag set
- The distance between the long stretch start point and the UAV's estimated position is less than *accepted_distance_to_start_point*
- The UAV's estimated velocity is parallel to the wind velocity used to calculate the initial CARP from State 2. The parallelism must be accurate to within *percent_accurate* percent

When these statements are true, the UAV's position and velocity is assumed to fulfil the requirements described in Section 2.5.1 to start approaching the target.

If *direct_to_opt* is set to *true* in the .ini file, this state is avoided, making the UAV go directly to the end point. This only works with the path planners using optimization methods: OWSI or the optimal path planner.

State 4: Go to End Point

This state's purpose is to reach the CARP with the required velocity. This is achieved using different approach methods described in Section 2.4 to 2.5. In general, this state ends either when the CARP or long stretch end point has been reached. In the case where the CARP has been reached, *glide_time* and *drop_time* are taken into account by estimating where the UAV will be in that amount of time, such that the propeller can be turned off before the drop takes place. To see if the CARP has been reached, the method in 2.6 has been used. The next state is then State 5.

In the case where the long stretch end point has been reached, the CARP is considered as missed, and the mission is viewed as a failure. The next state is in this case State 6.

State 5: Glide and Drop Beacon

In this state, no new updates are made to the UAV's current path. The UAV will glide until it is as close to the CARP as possible. This means either closer than s/r, where s is speed and r is state estimate update rate, or that the newest calculated distance to CARP is further away than the previous calculated distance to CARP (for more, see Section 2.6). The UAV will then drop the beacon and glide for one more period of *glide_time* to avoid hitting the payload with the propeller.

State 6: Go to Safe Height and Loiter

When the final glide is finished, the UAV rises in a helix to a safe height specified as *safe_height*. When *FollowRefState.proximity* has a high bit in *PROX_Z_NEAR* and *PROX_XY_NEAR*, the UAV goes back into State 1.

4.5 Software Installation and Setup

Some off-the-shelf software has been described in Section 4.1 and 4.2. These programs had to be installed and set up appropriately to work with this project's hardware. This is presented in the following sections.

4.5.1 Installation on the X8

All software, other than firmware, installed in the X8 was installed on the BBB and the PWM generator. The PWM generator was simply programmed using Atmel Studio 6.2, but the BBB was more challenging. First, the OS was installed. As mentioned in Section 4.1.2, GLUED was decided to be the UAV's OS of choice. To install GLUED, the instructions made by LSTS was followed (GLUED, 2014). Later, DUNE was acquired from git, compiled and transferred to the BBB. The procedures are described in more detail below.

GLUED on the BeagleBone Black

When installing GLUED for the BBB, one must first build GLUED, then transfer it to a memory device and then boot from the memory device on the embedded computer. The installation steps are provided in Appendix D. After following the guide the BBB is booting to GLUED when powered.

Cross Compilation of DUNE for the BeagleBone Black

Cross compilation of DUNE was the next natural step. This was achieved by using the toolchain created when building GLUED. The procedure used for cross compilation is given in Appendix E.

To test whether the cross compilation was successful or not, the resulting program can be tried to run on the computer used for cross compilation after the second step described in the appendix. It should not run. If it does, that means it is compiled for the local computer and not for the embedded device.

This concludes the installation of DUNE on the payload computer, BBB. To run DUNE on the BBB, a .ini file with the same name as the system file mentioned in Appendix D (ntnu_x8_006 for this project) must be available in the DUNE folder when running the command "make package". An example of the .ini file used in this thesis can be found in Appendix H.

4.5.2 Installing Software on the Ground Station

Installing the required software on the ground station was quite trivial compared to installing the BBB's software. Neptus is available as a package to be installed on Linux or Mac in the develop branch of https://github.com/LSTS/neptus.git. It is installed in Linux using the *./ant* command, while it is available as an executable file on Windows.

APM:Planner 2.0 and Mission Planner are also readily available for direct installation via packages or executables at APM Planner 2.0 (2014).

Chapter 5

Simulations

Simulation was a great tool for developing software that was needed in this thesis. It was also useful for gathering results without flying, and therefore make relatively large amounts of data available. To evaluate the data, accuracy and precision is used. The terms are based on the normal distribution model, which is assumed to apply to the measurements. Accuracy is defined as the mean error, while precision is depending on the standard deviation. Normally, the terms Mean Radial Spherical Error (MRSE) and Distance Root Mean Squared (DRMS) are used to evaluate precision in respectively three and two dimensions (GPS Position Accuracy, 2003).

Several tests are presented through this chapter.

Controller Tuning, Section 5.1.1 This section explores the stability and the tuning of the LOS and SMC controller in a SIL environment.

Wind Estimation, Section 5.1.2 Two scenarios are simulated in SIL to test the wind estimator accuracy and precision.

CARP Approach Method Accuracy, Section 5.1.3 As specified in Section 1.6, this thesis aims for hitting sea ice with a 10 m radius 95% of the time. These tests are designed to see how close to the CARP the UAV can drop in SIL.

Simulation of Free Falling Sphere for Varying Winds, Section 5.2 When the beacon is released, the trajectory depends on the beacon's initial velocity and the wind affecting it. This test will see how much the beacon's trajectory is changed by wind estimation errors.

5.1 Software in the Loop Simulations

To gain som knowledge of the path planners, flight controllers, release accuracy and wind estimation accuracy before flight tests were performed, SIL testing was used. Simulations using SIL was also useful for debugging while programming. JSBsim (Section 4.2.4) and Ardupilot SITL simulator (Section 4.2.3) was used to test DUNE (Section 4.1.3 and 4.4) software in SIL. How these programs communicate can be seen in Figure 5.1. Neptus (Section 4.2.1) was used for overview and sending commands to DUNE.

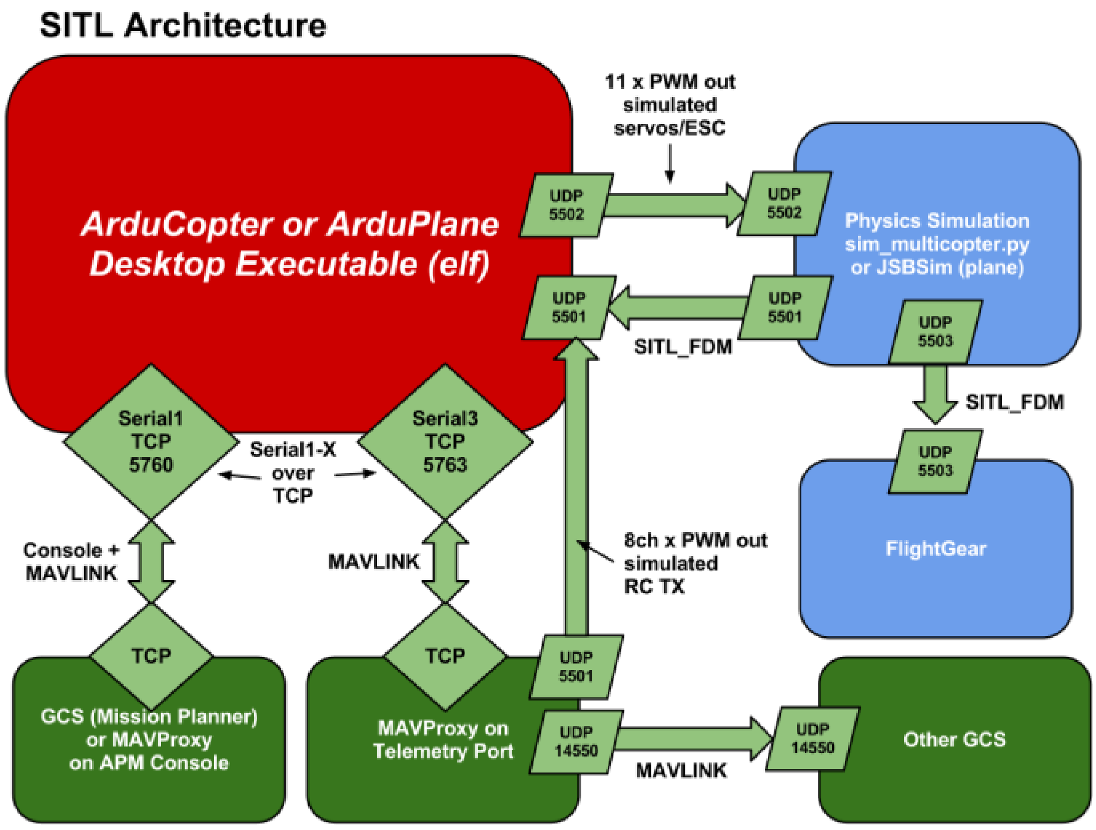

SITL Architecture
ArduCopter or ArduPlane Desktop Executable (elf)
11 x PWM out simulated servos/ESC
UDP 5502
UDP 5502
Physics Simulation sim_multicopter.py or JSBSim (plane)
UDP 5501
UDP 5501
SITL_FDM
UDP 5503
SITL_FDM
UDP 5503
Serial1 TCP 5760
Serial1-X over TCP
Serial3 TCP 5763
Console + MAVLINK
MAVLINK
8ch x PWM out simulated RC TX
FlightGear
TCP
TCP
GCS (Mission Planner) or MAVProxy on APM Console
MAVProxy on Telemetry Port
UDP 5501
UDP 14550
MAVLINK
UDP 14550
Other GCS

Figure 5.1: SIL software overview, from SITL Simulator

5.1.1 Controller Tuning

This section describes tuning of the LOS controller from Section 2.4.

To get significant SIL approach method accuracy test results, the controllers needed to be tuned to the specific SIL environment. The UAV in SIL has different flight characteristics from the X8. In a way, the SIL tuning became practice for the future, as the controllers had to be re-tuned in the field before future flight tests could be started. The tuning was performed by setting a four point path for the UAV to follow. The autopilot's native bank controller was already well tuned. The only other parameter to be tuned was the look-ahead gain. Many values were tested, and the resulting path was evaluated. The best possible look-ahead distance was found to be 50 m, the same results as in Fortuna and Fossen.

5.1.2 Wind Estimation

To tune the wind estimator described in 2.3, the wind was set to be 2 and 5 m/s northward during two recorded simulated flights. The Q and R values were changed beforehand until the measurement settled sufficiently fast.

Then the size of the observability gramian was investigated during the start of the flight. When the wind estimate had settled at about the reference wind speed, the observed gramian size (the ϵ in Johansen et al. (2015)) was 0.001, and the measurement was then considered as trusted.

The wind estimator is depending on a constantly changing attitude for the system to be observable. During simulations, it was observed that the estimator was quite accurate when the attitude was changing, but falls when the UAV is travelling in one direction.

The accuracy is compared to that of the Pixhawks own estimator in Figure 5.2 to 5.5. Simulator settings can be found in Table 5.1, while the mean and variances of the estimations are found in Table 5.2 and 5.3. Plots, mean values and variables are made using the MATLAB script found in Appendix I.4 and I.5.

Type	Value
Enabled	Always
Entity Label	Wind Estimator
Sample Window Size	1000
Q Size	700
Trusted Limit	1e-3

Table 5.1: SIL wind test variables

dlim

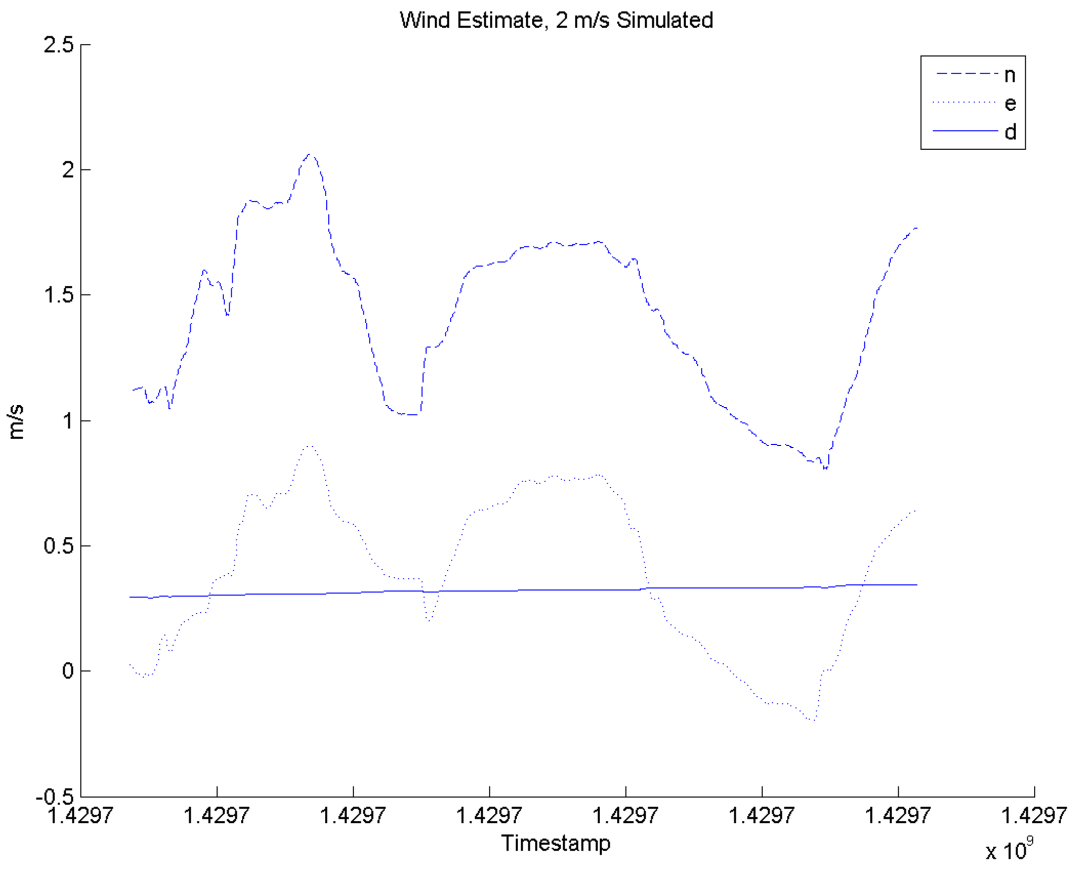


Figure 5.2: Wind estimation in simulation with 2 m/s wind

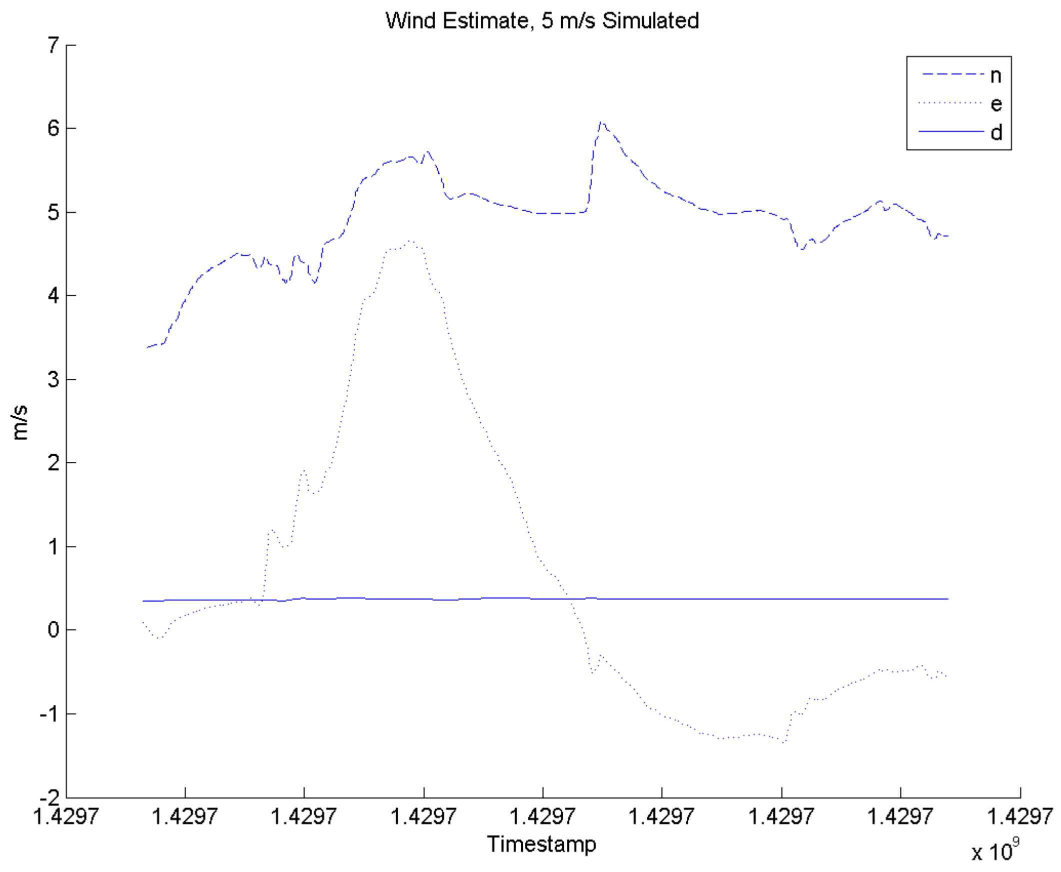


Figure 5.3: Wind estimation in simulation with 5 m/s wind

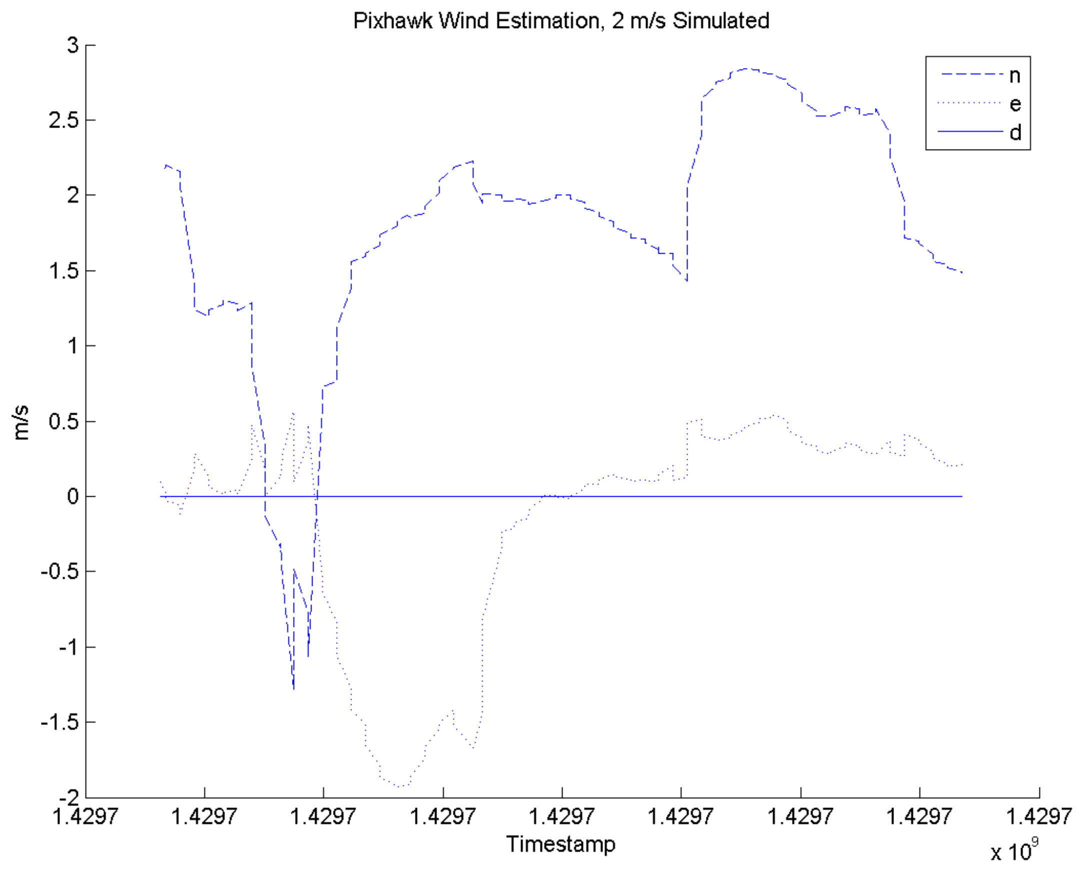


Figure 5.4: Pixhawk wind estimation in simulation with 2 m/s wind

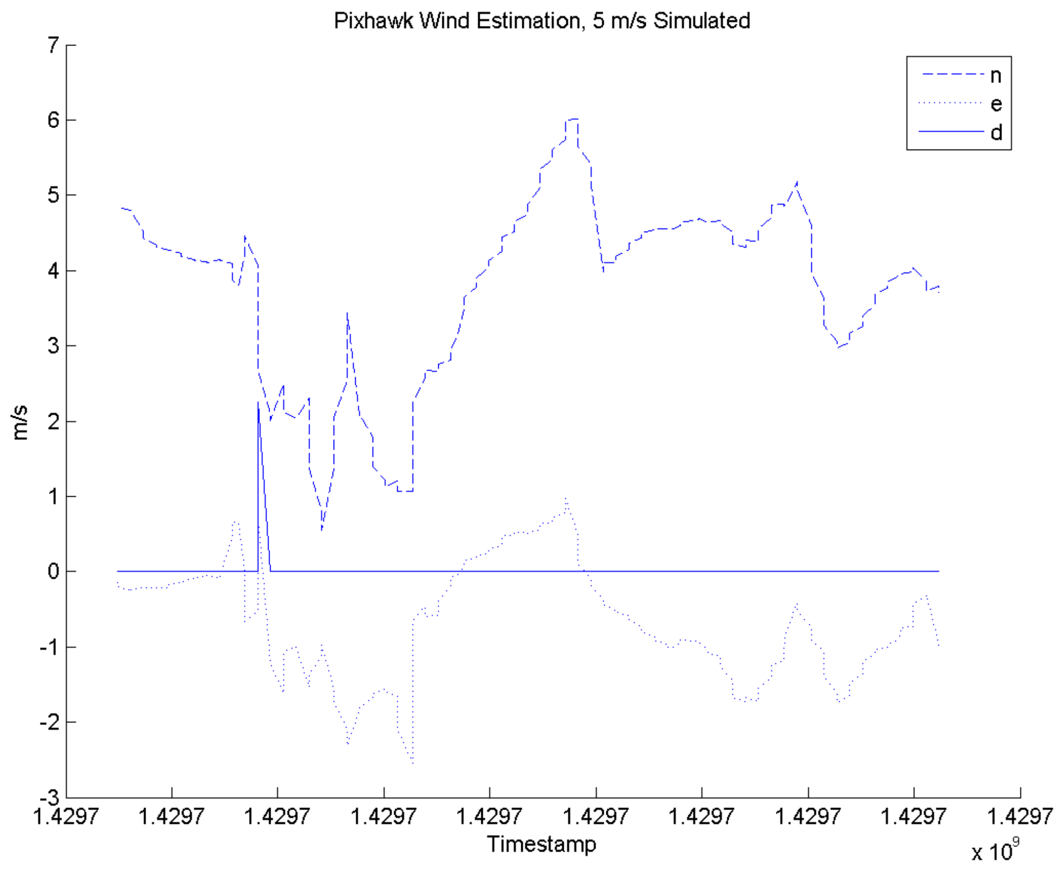


Figure 5.5: Pixhawk wind estimation in simulation with 5 m/s wind

Type	**Direction**	**Value**
Mean error	North	0.59 m/s
	East	-0.39 m/s
	Down	-0.32 m/s
Error variance	North	0.11 m/s
	East	0.09 m/s
	Down	1.77e-4 m/s

Table 5.2: Errors and variance for wind estimation, 2 m/s wind

Type	**Direction**	**Value**
Mean error	North	0.093 m/s
	East	-0.61 m/s
	Down	-0.37 m/s
Error variance	North	0.28 m/s
	East	3.11 m/s
	Down	$6.77e^{-5}$ m/s

Table 5.3: Errors and variance for wind estimation, 5 m/s wind

Type	**Direction**	**Value**
Mean error	North	0.24 m/s
	East	0.15 m/s
	Down	0 m/s
Error variance	North	0.72 m/s
	East	0.54 m/s
	Down	0 m/s

Table 5.4: Errors and variance for Pixhawk wind estimation, 2 m/s wind

Type	**Direction**	**Value**
Mean error	North	1.24 m/s
	East	0.65 m/s
	Down	-0.016 m/s
Error variance	North	1.59 m/s
	East	0.60 m/s
	Down	0.036 m/s

Table 5.5: Errors and variance for Pixhawk wind estimation, 5 m/s wind

5.1.3 CARP Approach Method Accuracy

Many different approach methods were used for these SIL tests: OWSI, Optimal and LS path planners with the waypoint and LOS with SMC controllers. The ISL path planner was not tested, as it would in theory be the same as the LS path when using a LOS controller, and the waypoint controller was quickly discarded. .ini variables from 4.4.3 can be found in Table 5.6. All other .ini entries were as in Appendix H. All approach strategies were initiated by travelling up against the wind (2 m/s northward) and the UAV speed set point was 17 m/s.

SIL Test 1: Waypoint Controller Compared to LOS and SMC Controller

The first test in this section shows the difference in accuracy and precision between the waypoint controller and the LOS with SMC controller by studying 10 samples of LS path using either controller. The results are shown in Table 5.7 and 5.8.

Type	**Value**
Altitude Interval	1
Connection Timeout	3
Distance To LSSP	100
Start Point Distance From CARP	200
End Point Distance From CARP	400
Percent Accurate	7
Radius	100
Release Height	50
Speed	17
Drop Error	50
Guidance Mode	[DEPENDS ON TEST]
Safe Height	100
Max Drop Current	9
Glide Time	2
Drop Time	0.6
Use Bank Controller	true
Use Wind Estimator	true
Step Size	0.001
Optimize Points	30
Optimize Rads	2
Direct To Optimal	false
Max Counter	100000
Optimize Once	false
Optimation Rate Inverse	10
Position Weight	15
Velocity Weight	1
OWSI Min Distance	20

Table 5.6: Autonomy.DropOnTarget variables

Approach Strategy	**CARP Distance [m]**	V_e **north [m/s]**	V_e **east [m/s]**	V_e **down [m/s]**
Long Stretch	4.80	0.83	0.32	2.33
LOS + SMC	8.92	1.21	-0.65	-0.33
	6.32	2.42	-1.33	1.24
	0.81	2.45	-3.82	1.32
	0.65	2.17	-1.64	0.62
	5.52	3.75	0.49	0.39
	4.38	2.43	-0.77	-0.22
	8.98	-0.20	-0.22	0.22
	13.92	2.16	-0.51	1.43
	10.31	1.92	0.05	0.78
	3.36	2.93	-0.26	1.70
Absolute error	μ: 6.32, σ^2: 18.13	μ: 2.16, σ^2: 0.91	μ: 0.97, σ^2: 1.24	μ: 0.83, σ^2: 0.31

Table 5.7: SIL approach method accuracy measurements for the LS path planner with LOS and SMC controller where V_e is velocity error, μ is the mean and σ is the standard deviation

Approach Strategy	**CARP Distance [m]**	V_e **north [m/s]**	V_e **east [m/s]**	V_e **down [m/s]**
Long Stretch	30.36	1.40	2.51	-0.03
Waypoint	13.32	1.26	0.22	-1.56
	13.70	1.51	2.93	-0.45
	9.00	0.57	0.85	0.42
	13.39	-0.28	-1.24	0.50
	41.56	1.10	1.44	-0.42
	0.82	1.15	2.53	0.19
	16.71	2.76	5.14	-0.93
	13.84	0.53	0.36	-0.39
	21.95	-0.36	-2.98	-0.38
Absolute error	μ: 17.47, σ^2: 130.82	μ: 1.09, σ^2: 0.54	μ: 2.02, σ^2: 2.25,	μ: 0.51, σ^2: 0.17

Table 5.8: SIL approach method accuracy measurements for the LS path planner with waypoint controller where V_e is velocity error, μ is the mean and σ is the standard deviation

SIL Test 2: Long Stretch, Optimal and OWSI Path Planners with LOS and SMC Controller

The second test provides extensive results on how accurate the three different path planners are while using the LOS and SMC controllers. The waypoint controller was not used as its accuracy was much lower than the LOS in the previous test. 500 samples were taken of each approach method with details on actual drop point compared to CARP. Resulting mean values and variances are shown in Table 5.9 to 5.11, while plots of the release points, accuracy and DRMS are shown in Figure 5.6 to 5.10. Measurements and plots are given in body frame and DRMS is defined as a circle containing 65% of release points (GPS Position Accuracy, 2003). The MATLAB scripts used to make calculations and plots are found in Appendix I.6, I.7 and I.8.

Approach Strategy	**Measurement**	**Direction**	**Value**
LS, LOS controller	Mean position error	x	0.69
		y	2.16
		z	-1.63
	Position error variance	x	3.17
		y	5.14
		z	2.66
	Mean velocity error	x	-0.37
		y	0.64
		z	0
	Velocity variance	x	0.68
		y	0.58
		z	0

Table 5.9: SIL accuracy for LS path planner with LOS controller

Approach Strategy	**Measurement**	**Direction**	**Value**
Optimal, LOS controller	Mean position error	x	-1.37
		y	0.27
		z	-0.46
	Position error variance	x	3.53
		y	2.86
		z	2.31
	Mean velocity error	x	-0.13
		y	0.60
		z	0
	Velocity variance	x	0.45
		y	0.56
		z	0

Table 5.10: SIL accuracy for Optimal path planner with LOS controller

Approach Strategy	**Measurement**	**Direction**	**Value**
OWSI, LOS controller	Mean position error	x	-1.76
		y	0.15
		z	-0.49
	Position error variance	x	5.63
		y	7.18
		z	3.14
	Mean velocity error	x	-0.16
		y	0.58
		z	0
	Velocity variance	x	0.68
		y	0.64
		z	0

Table 5.11: SIL accuracy for OWSI path planner with LOS and SMC controller

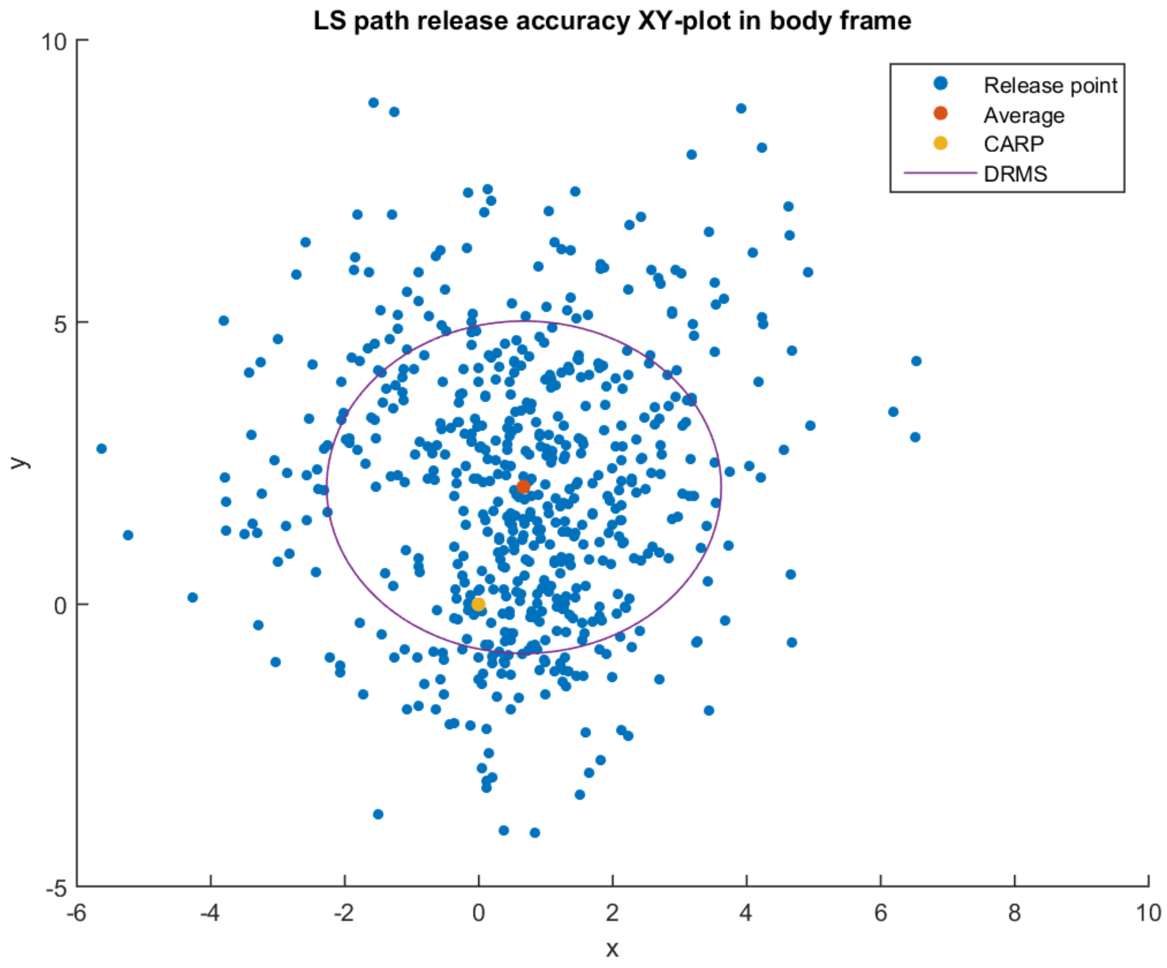


Figure 5.6: Drop accuracy for LS with LOS

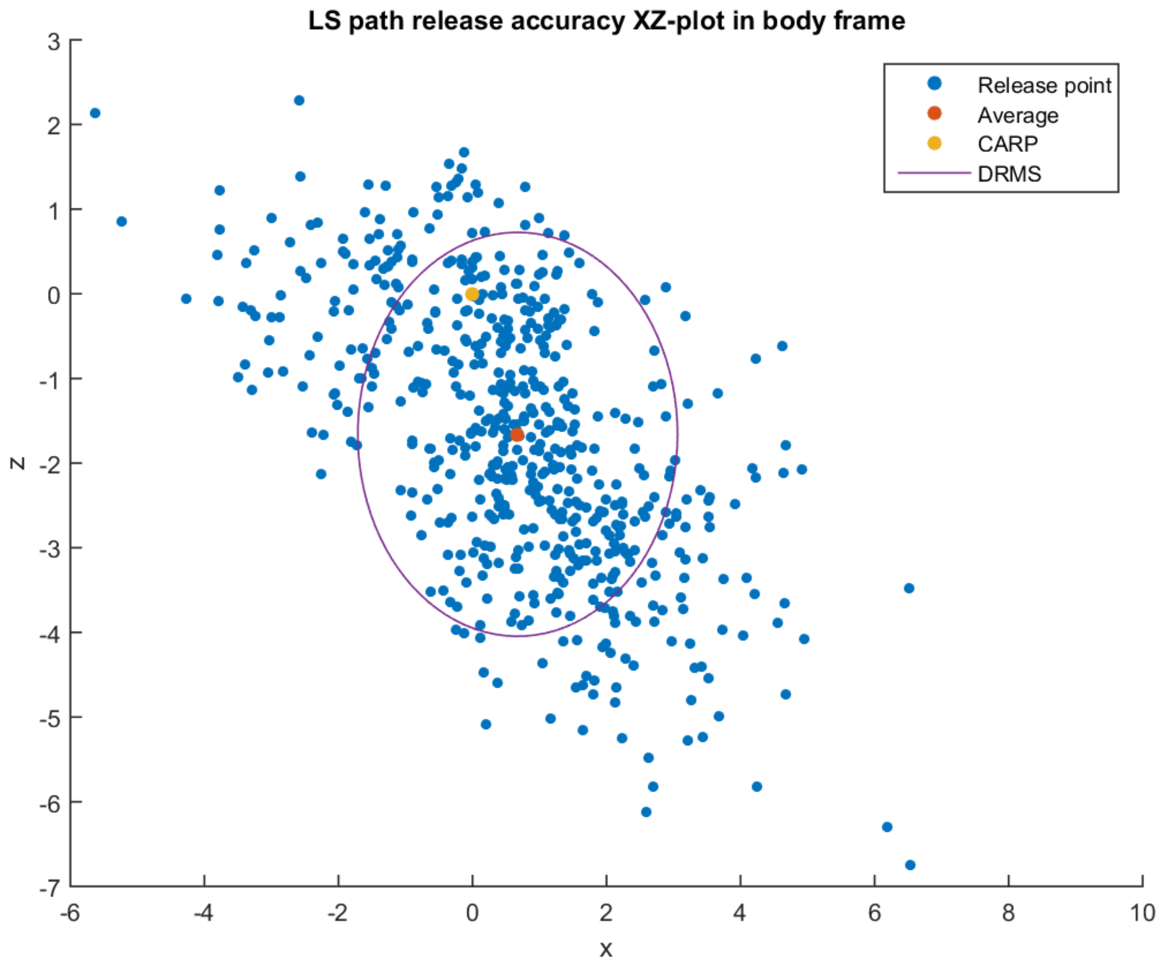


Figure 5.7: Drop accuracy for LS with LOS

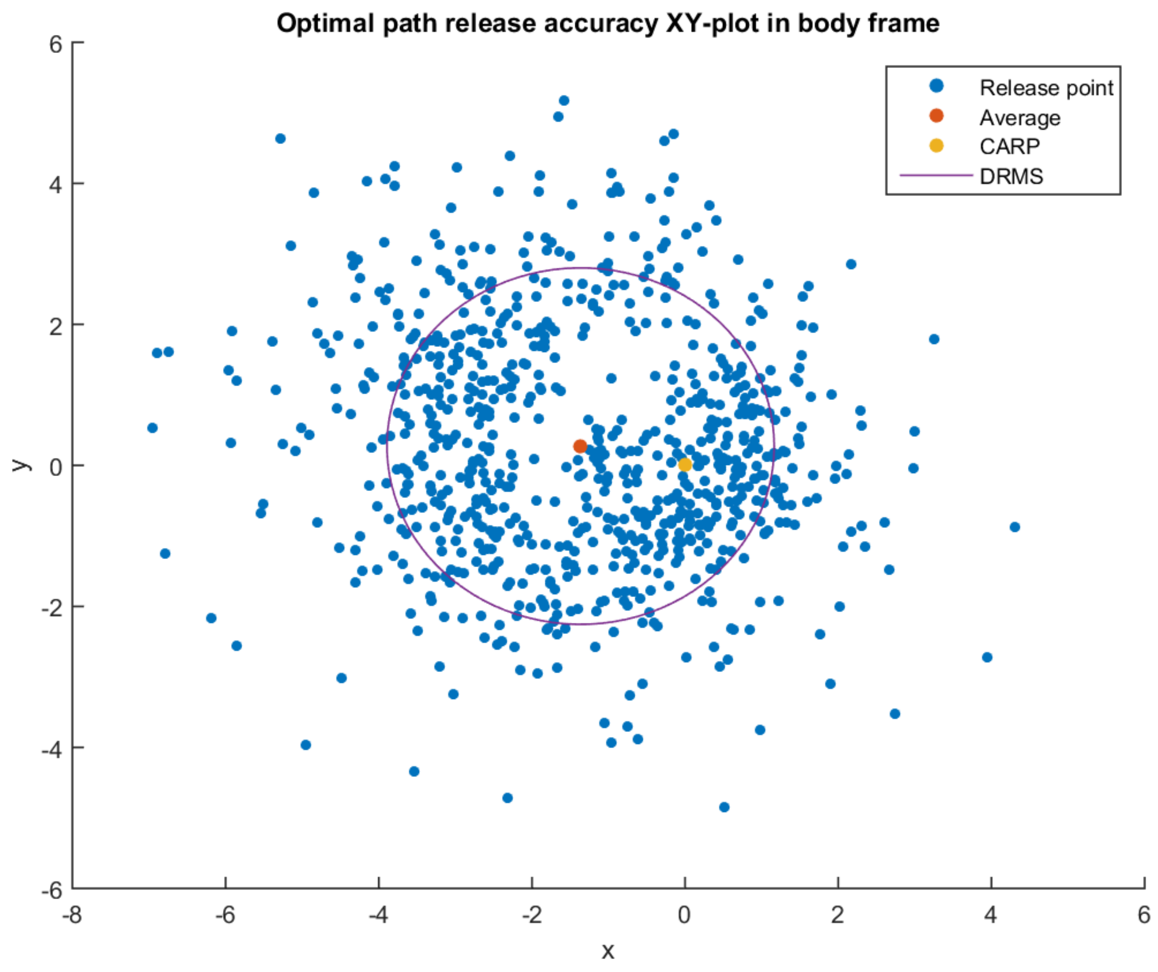


Figure 5.8: Drop accuracy for Optimal path with LOS

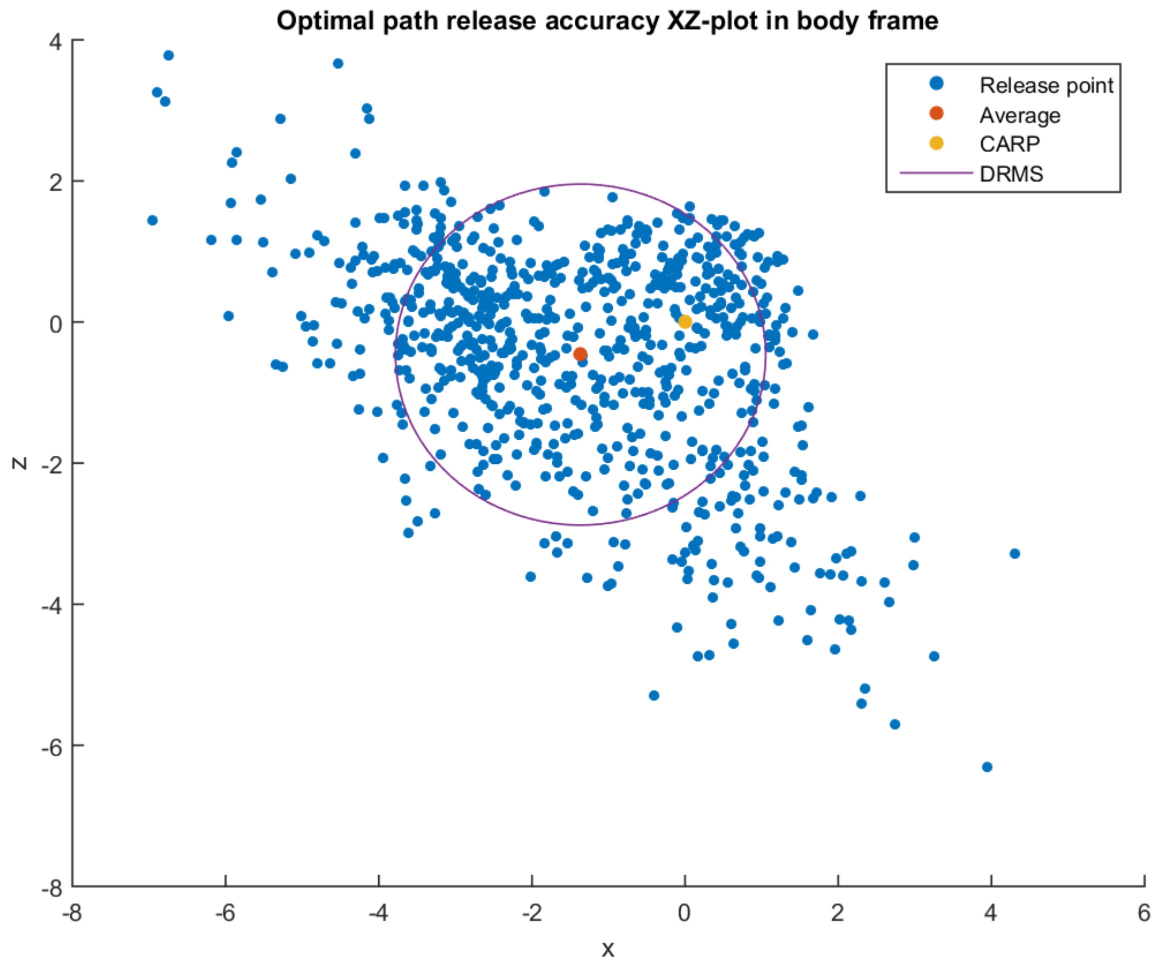


Figure 5.9: Drop accuracy for Optimal path with LOS

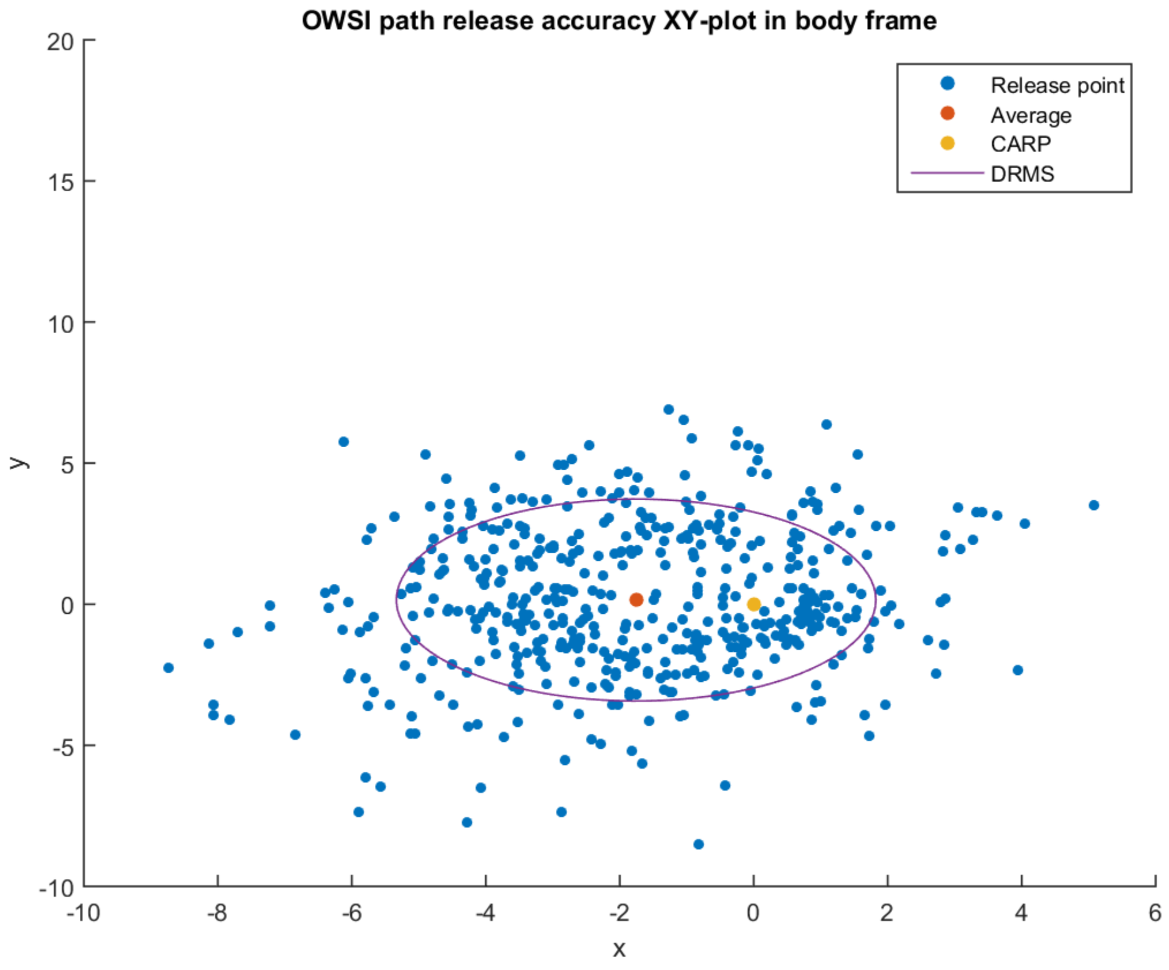


Figure 5.10: Drop accuracy for OWSI with LOS

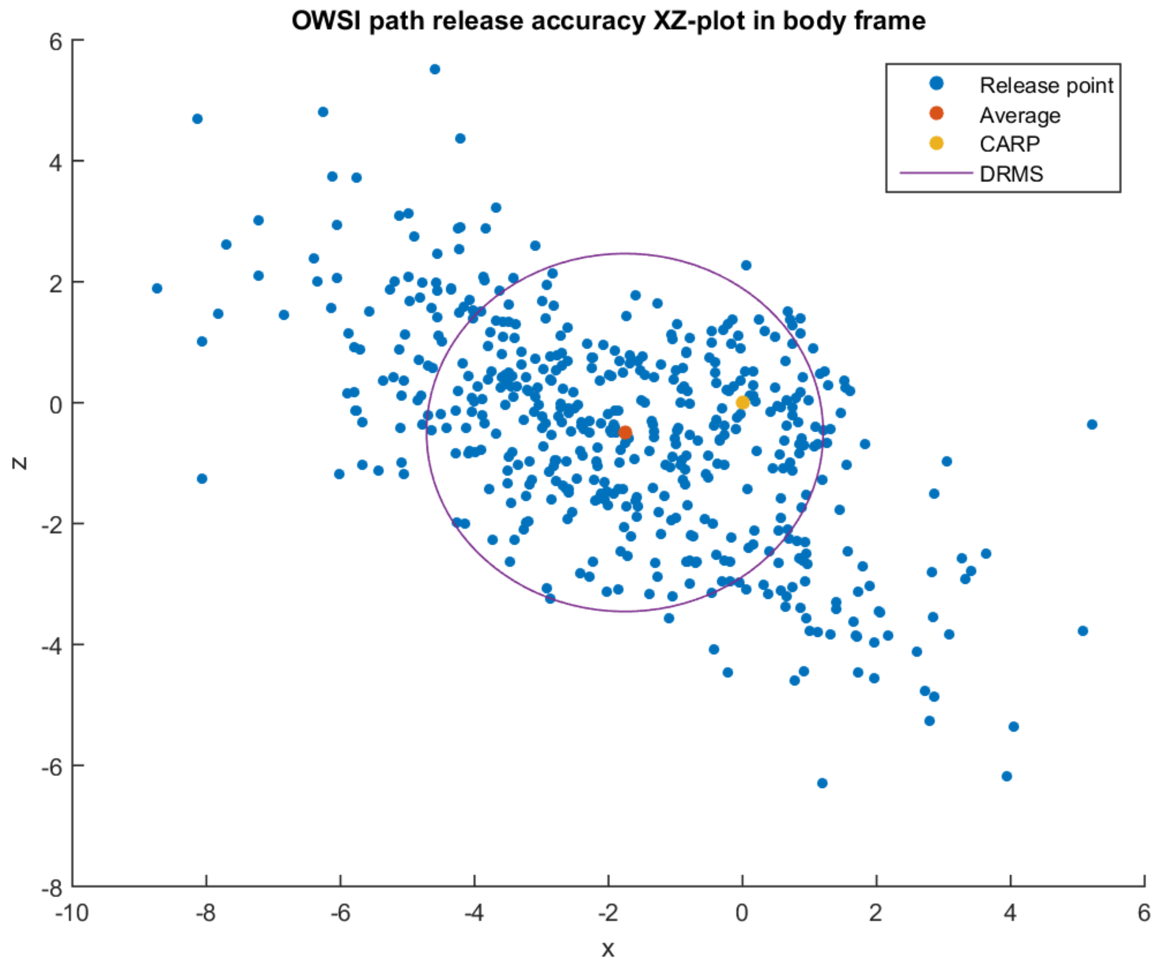


Figure 5.11: Drop accuracy for OWSI with LOS

5.1.4 SIL Conclusion

SIL testing has given some important results showing that the expected wind estimate and release accuracy are quite good.

The wind estimator ended up being accurate to

5 m/s wind	Accuracy	Mean error	$\sqrt{0.093^2+0.61^2+0.37^2}=0.72$ m/s
	Precission	MRSE	$\sqrt{0.28+3.11+6.77e^{-5}}=1.84$ m/s
2 m/s wind	Accuracy	Mean error	$\sqrt{0.59^2+0.39^2+0.32^2}=0.78$ m/s
	Precission	MRSE	$\sqrt{0.11+0.095+1.77e^{-4}}=0.45$ m/s

where MRSE is the sphere containing 61% of the measurements (GPS Position Accuracy, 2003).

Comparing Table 5.4 to 5.2 and Table 5.5 to 5.3 shows that the Pixhawk wind estimator is a bit more accurate, but that the wind estimator from Johansen et al. (2015) is much more precise.

In the first approach method test, the waypoint controller was used with LS path planner, and the results in Table 5.8 are comparable to the results from LS path planner with the LOS controller in Table 5.7. The results show that the LOS controller gives much better accuracy with 6.310 m mean and 18.13 m variance against 17.465 m mean and 130.82 m variance.

The approach methods had an absolute position and velocity error as shown in Table 5.12 while using LOS and SMC controller. It was also observed during testing that the approach strategy using the OWSI path planner oscillated less than the approach strategy using the Optimal path planner. Another observation, this time from the measurements, is that the z velocity error was always zero. That this is true is improbable and should be investigated in future work.

Path	**Type**	**Accuracy (Mean Error)**	**Precision (MRSE)**
LS	Position	$\sqrt{0.69^2+2.16^2+1.63^2}=2.79$ m	$\sqrt{3.17+5.14+2.66}=3.31m$
	Velocity	$\sqrt{0.37^2+0.64^2+0^2}=0.74$ m/s	$\sqrt{0.68+0.58+0}=1.23$ m/s
Optimal	Position	$\sqrt{1.37^2+0.27^2+0.46^2}=1.47$ m	$\sqrt{3.53+2.86+2.31}=2.95m$
	Velocity	$\sqrt{0.13^2+0.60^2+0^2}=0.61$ m/s	$\sqrt{0.45+0.56+0}=1.01$ m/s
OWSI	Position	$\sqrt{1.76^2+0.15^2+0.49^2}=1.83$ m	$\sqrt{5.62+7.18+3.14}=3.99$ m
	Velocity	$\sqrt{0.15^2+0.57^2+0^2}=0.60$ m/s	$\sqrt{0.68+0.64+0}=1.15$ m/s

Table 5.12: SIL precision and accuracy for approach strategies

5.2 Simulation of Free Falling Sphere for Varying Winds

Calculations of the ballistic paths for a free-falling object have several uncertainties, such as wind magnitude and direction (and wind gust). Utilizing MATLAB to perform simulations of ballistic paths for a free-falling sphere and the state-space model in Section 2.1, the ballistic paths and probability distribution of the impact zone can be calculated. These simulations are based on the work done in Fuglaas (2014) and is a repetition of the experiment.

The sphere used in this simulation has a mass off 200 g and a cross-sectional area off $0.05^2\pi$ m^2, which should be a proper estimate of the beacon. An air density of $\rho = 1.225\ kg/m^3$ (Air density at $15°C$ (Fossen, 2011)) is used. Also, using wind velocities of 5 m/s with an uncertainty of 50%, making the wind varying from light breeze to a moderate breeze (Fossen, 2011), and wind direction fluctuating about 180 (±10 degrees) would produce a fairly good uncertainty distribution in the wind estimate. Using a normal distribution for estimating the wind uncertainties with mean $\mu = \pi$ and standard deviation of $\sigma = 1$, the ground impact point uncertainties were estimated. To solve, MATLAB's ode45 solver was used with a time step of 0.01 s. Figure 5.12 show the spread of the ballistic paths for varying winds and Table 5.13 for simulation settings.

Type	**Value**	**Description**
A	$\pi 0.05^2[m^2]$	Cross-sectional area of sphere
C_D	0.5	Drag constant for the sphere
m	$0.200[kg]$	Mass
ρ	$1.225[kg/m^3]$	Air density at $15°C$
g	$9.81[m/s^2]$	Gravity
w	$5[m/s] \pm 50\%$	Wind speed
$\angle w$	$180° \pm 10°$	Wind direction angel

Table 5.13: Physical properties of free fall simulation

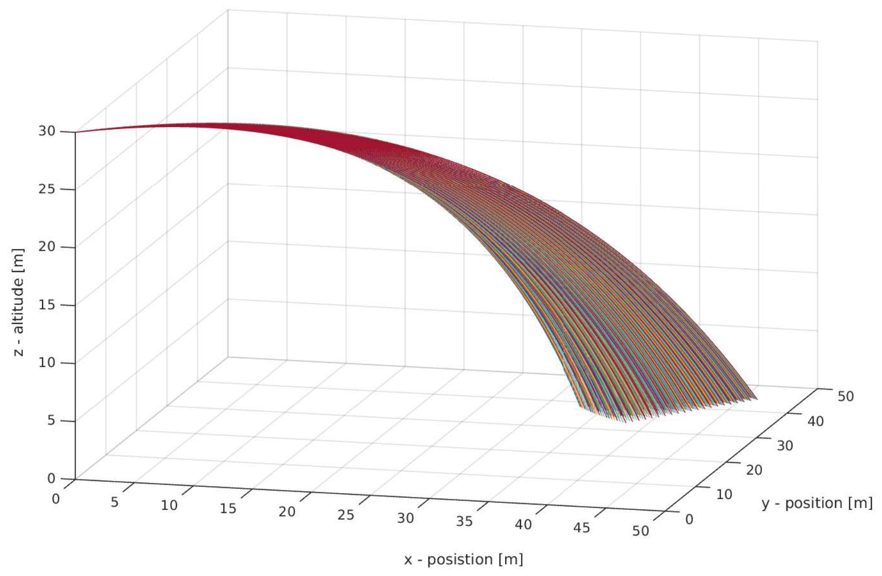


Figure 5.12: Multiple ballistic paths of sphere dropped from 30 meter, using wind velocity 5 m/s ±50% and wind angel 180° ± 10°

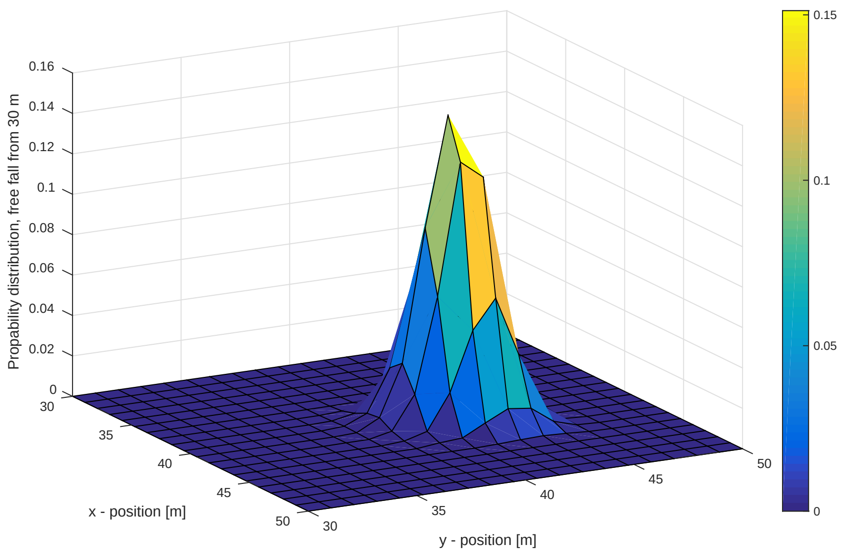


Figure 5.13: Ground impact probability distortion of sphere drop from 30 meter, using wind velocity 5 m/s ±50% and wind angel 180° ± 10°

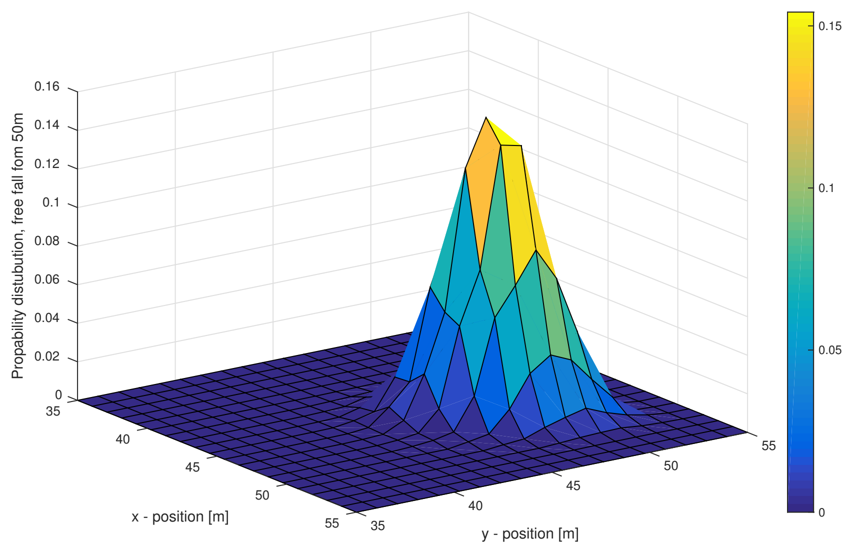


Figure 5.14: Ground impact probability distortion of sphere drop from 50 meter, using wind velocity 5 m/s ±50% and wind angel 180° ± 10°

5.2.1 Simulation Conclusion

As seen from the simulations in Figure 5.13 and Figure 5.14, the release height, as well as the wind factor, is a major contributor to the distribution of the ground-impact uncertainty.

Chapter 6

UAV and Payload Testing

When the hardware was set up, tests were conducted to verify that the system was working correctly. These tests are described in Section 6.1. Flight tests were then performed and they are described in Section 6.2. Several tests were performed in each of these sections.

BeagleBone Black Optimization Calculation Time, Section 6.1.1 Testing to see how much time the BBB uses for the optimization performed during calculation of OWSI or Optimal path.

Release Mechanism Test, Section 6.1.2 This test was devised to make sure the release mechanism was compatible with the BBB.

Sending Pixhawk Data Using IMC Over Radio Link, Section 6.1.3 Testing the radio link was important to avoid complications during flight testing.

Full Payload Test, Section 6.1.4 The final test before flight test could commence. All payload equipment was tested at once by mimicking a flight.

State Estimation Accuracy and Precision Test, Section 6.2.1 By flying with a hexacopter carrying a Pixhawk and an RTK GPS, the Pixhawk's state estimation was tested. This had to be quite accurate for the project to maintain the 10 m accuracy required in Section 1.6.

Approach using Straight-Line Path and Waypoint Controller, Section 6.2.2 The approach method was tested without releasing a beacon to see how close the UAV would get to the CARP in flight tests. The waypoint controller was used, and the accuracy was required to be better than the 10 m mentioned in 1.6.

Approach using Incremented-Straight-Line Path and Waypoint Controller, Section 6.2.3 The approach method was tested without releasing a beacon to see how close the UAV would get to the CARP in flight tests. The waypoint controller was used, and the accuracy was required to be better than the 10 m mentioned in 1.6.

6.1 Hardware Tests

Before flight tests could be performed, every subsystem was tested to show that they worked sufficiently. The first tests contained in this section were designed to test different hardware subsystems, while in the final hardware test, the complete system was tested.

6.1.1 BeagleBone Black Optimization-Calculation Time

The optimization used in the Optimal (Section 2.5.3) and OWSI (Section 2.5.4) path planners is quite computationally demanding. To find out how much time the BBB uses to perform the optimization, a test was conducted. The test consisted of running many optimizations on the BBB. Optimization parameters were: a time step of 0.001 s, 50 m release height and using 30 points in the RoF. All calculations were completed in between 60 and 100 ms.

6.1.2 Release Mechanism Test

This test was designed to test DUNE's compatibility with the BBB and the ability for DUNE to map GPIO output on the BBB. The test would also reveal whether the PWM generator and drop mechanism would work as intended.

A simple DUNE task was implemented which would set one GPIO pin high for two seconds and then low for five seconds. The physical manifestation of this would be that the drop

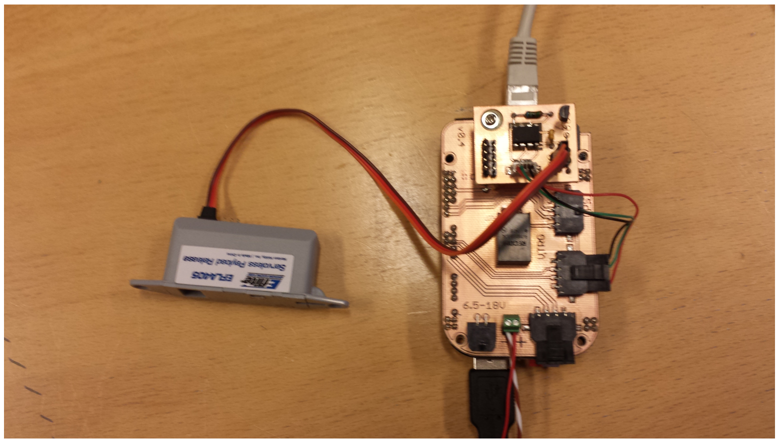


Figure 6.1: BBB with cape and PWM generator on top, wired to the drop mechanism on the left

mechanism would execute for two seconds and then wait for five seconds. See Figure 6.1 for illustration, and Appendix G for the code used in the test.

The release mechanism reacted as predicted, opening for two seconds and pausing for five.

6.1.3 Sending Pixhawk Data Using IMC Over Radio Link

To test the serial communication between the Pixhawk and the BBB, and to test IMC communication via the Rocket M5/Nanostation M radio link, a test was performed.

The hardware was set up as a ground station communicating with a UAV payload computer (the BBB), where the BBB used serial communication to communicate with the Pixhawk and Ethernet to communicate with the Rocket M5 and ground station. The ground station ran Neptus, while the BBB ran DUNE with two tasks. One task to communicate with the Pixhawk and one task to send the acquired autopilot data to the IMC bus. The ground station would then catch the autopilot data from the IMC bus and display it in Neptus. This would prove a working connection between the ground station and the BBB, and that a functional serial communication between the BBB and the Pixhawk existed. See Appendix G for the code used in the test, and Figure 6.2 and 6.3 for pictures.

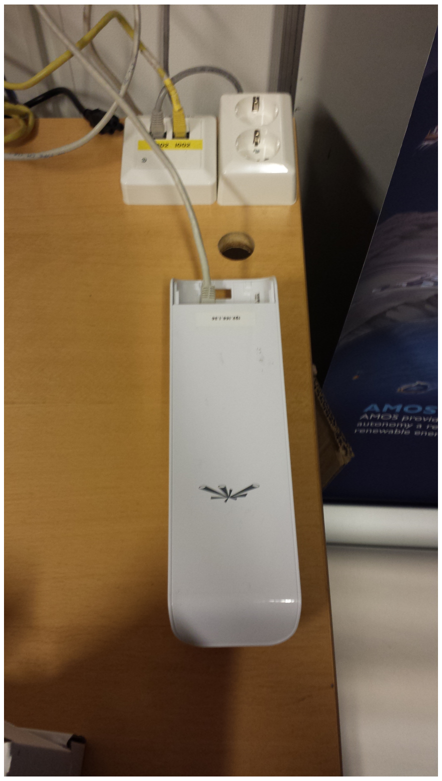


Figure 6.2: Nanostation M connected to the internal network on the ground station

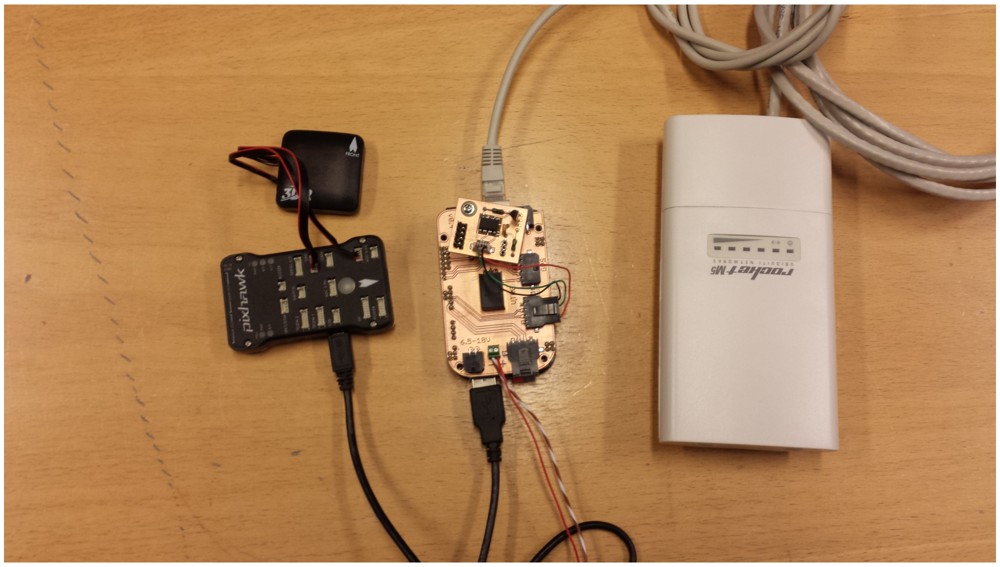


Figure 6.3: BBB and Pixhawk ready for testing with Rocket M5

The autopilot data was transferred successfully to the ground station, with so little lag that it was to be considered "live". The data link was functioning as expected.

6.1.4 Full Payload Test

To test the ground station and payload configuration without spending flight time, a simple test was proposed. The payload was placed in a plastic box and walked around on an airfield to simulate normal working conditions for the UAV and ground station.

The GPS signal was lost at times due to being held too close to a body with an arm on top. When this was discovered, the GPS antenna was moved outside of the box and as far away from the carrying person as possible. The signal promptly returned.

IMC messages were recorded accurately and reliably, and it was concluded that the payload worked well.

6.1.5 Hardware Test Conclusion

After performing the tests in Section 6.1.1 to 6.1.4, all systems were shown to work reliably. This made the research team feel secure about conducting flight tests.

6.2 Flight Tests

The results of this thesis are heavily hinged upon flight tests. Unfortunately, flight tests using the LOS and SMC controllers were not performed, nor tests using the Optimal path planner or the OWSI path planner. One of the flight tests is pictured in Figure 6.4.

Figure 6.4: Flight testing

The tests that were performed are described in detail in the following sections.

6.2.1 State Estimation Accuracy and Precision Test

The state estimation was tested by using a hexacopter with both Piksi (Piksi RTK-GPS, 2014), an RTK-GPS, and Pixhawk on board. The RTK-GPS is so accurate that it is treated as the reference in this test with a precision of 2 cm horizontally and 6 cm vertically (Piksi FAQ, 2013). 1056 measurements were taken from DUNE tasks and were processed using the MATLAB script in Appendix I.3. Resulting plots are found in Figure 6.5 and 6.6.

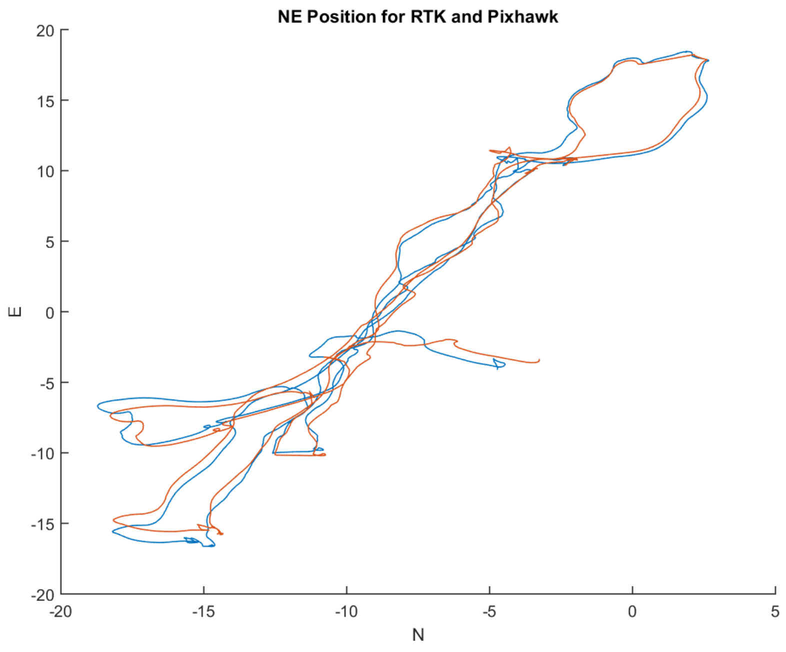


Figure 6.5: Plot of position in north east plane

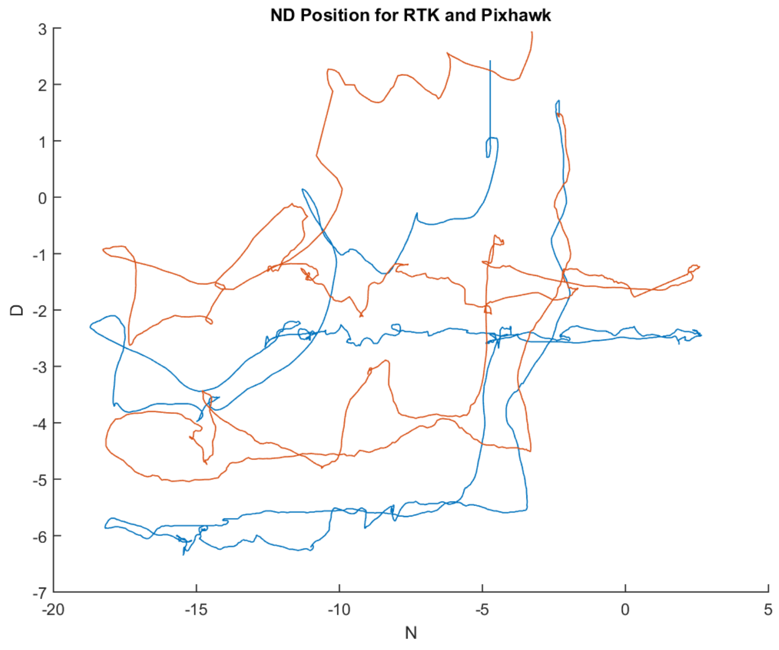


Figure 6.6: Plot of position in north down plane

Using MATLAB's *mean()* and *var()* on measurments taken at about the same time (the timestamps were compared), the Pixhawk's accuracy is listed in Table 6.1.

Type	Direction	Value
Mean error	North	-0.23 m
	East	-0.10 m
	Down	-1.36 m
Error variance	North	0.23 m
	East	0.33 m
	Down	0.40 m

Table 6.1: Errors and variance for Pixhawk state estimation

6.2.2 Approach using Straight-Line Path and Waypoint Controller

The first approach strategy that was tested was the LS path planner using the waypoint controller. A description of this approach can be found in Section 2.5.1 and 2.4.1. During this test the beacon placement was on the wings to test compatibility with the launch catapult.

Day one: The Drop-on-Target task was started while in the air at a 100 m altitude loiter. The beacons were not attached during any of the day's flights, as the author only wanted to test the state machine and approach accuracy, and not the drop mechanisms.

The test was started twice during the same flight, at 15.44.28 and 15.47.05, and was simulating a drop at 15.45.32, 1.04 after start, and at 15.47.51, 0.46 after start, at measured distances of 12.44 m and 15.17 m away from the CARP.

The wind was classified as a gentle breeze during testing and estimated to be about 3.4m/s when dropping. The temperature varied between -5 and -10 C.

Day two: Testing with the test beacons was performed on the second day at 50 meters altitude. The first takeoff was a failure as the added weight made the catapult to slow. The X8 took off at 10.33.02 after mounting more rubber tubing to the catapult. Initiation of the state machine took place at 10.39.06 after some initial flight tests by the pilot.

One of the test beacons fell off during the aforementioned flight tests for some unknown reason. It was later discovered that the drop mechanisms would open when DUNE was restarted on the BBB, and this was probably the reason. The other test beacon was stuck, and had to be pried off after landing.

The drop signal was sent at 10.39.51, 0.45 after start, at a distance of 15.94 m from the CARP.

The wind was classified as a breeze during testing and estimated to be about 5.71 m/s when dropping. The temperature varied between -5 and -10 C.

6.2.3 Approach using the Incremented-Straight-Line Path Planner and Waypoint Controller

The second approach method that was tested was the ILS path planner combined with the waypoint controller. This method is about the same as the LS path planner with a waypoint controller, but on the long stretch, when the plane is flying against the wind, several intermediate waypoints are chosen. This way, the approach becomes a bit more like LOS guidance. This is described in more detail in Section 2.5.2 During this test the beacon placement was at the wings.

Day one: The first test was performed without test beacons. It was initiated at 15.52.20 and ended at 15.54.42. It was considered a failure as it never left the first state where it tries to get downwind of the CARP.

The erroneous code was corrected the same evening. The problem was most probably too high speed while loitering, as the X8 never entered the correct loitering circle. Correcting for this led to good loitering circles in following tests.

Day two: At the end of the second day, three tests were performed. These were all performed without test beacons, as these fell off during DUNE restart. DUNE restart had to be performed to let the pilot do post take off tests.

Take off was performed at 14.36.53, and was just barely successful. The X8 almost crashed sideways, as the angle of the X8 with test beacons on the catapult was too high.

The first test started at 14.38.51, and simulated a drop at 14.39.34, 1:43 after start 6.31 m away from the CARP.

The second test started at 14.39.49, and simulated a drop at 14.40.34, 0:45 after start 14.85 m away from the CARP.

The third test started at 14.41.10, and simulated a drop at 14.41.52, 0:42 after start 8.23 m away from the CARP.

Wind was estimated to be 5.4 m/s.

6.2.4 Flight Test Conclusion

The Pixhawk's state estimates were tested, and proved to be quite accurate as seen in Table 6.2.

Accuracy	3D Mean Error	$\sqrt{0.23^2 + 0.10^2 + 1.36^2} = 1.39$ m
Precission	3D MRSE	$\sqrt{0.23 + 0.33 + 0.40} = 0.98$ m
Accuracy	2D Mean Error	$\sqrt{0.23^2 + 0.10^2} = 0.34$ m
Precission	2D DRMS	$\sqrt{0.23 + 0.33} = 0.75$ m

Table 6.2: Total errors, DRMS and MRSE for Pixhawk state estimation

All tests using the LS path planner and waypoint controller were considered successful. The UAV was tested in quite cold weather (-10 C), and had no problems at that temperature. The approach method's accuracy was tested as well, resulting in drop errors of 12.44 m, 15.17 m and 15.94 m, averaging in 14.52 m.

All tests performed on the second day using the ILS path planner and waypoint controller were also considered successful. The approach method's accuracy was tested, resulting in drop errors of 3.4 m, 14.85 m and 8.23 m, averaging in 8.27 m.

No velocity error was recorded during these tests, but additional knowledge gained from the tests was that placing the beacon on the wings could lead to failed take-offs.

Chapter 7

Discussion

The main experimental results of the project's work will be discussed here, and an investigation of sources of error will be performed. Design choices will also be looked into when discussing the thesis' evolution with a focus on accuracy and precision.

Section 7.1 discusses the simulations performed in Chapter 5. Section 7.2 discusses the testing of hardware from Section 6.1, while Section 7.3 discusses the flight tests in Section 6.2.

7.1 Discussion on Simulations

The SIL tests were a considerably important part of this thesis. Most of the development was based on the findings in SIL. This means that the accuracy of the SIL environment compared to real world was very important.

There are differences between real world flight and SIL tests. JSBsim (Section 4.2.4) was used for physics simulations, which means that JSBsim's errors compared to actual flight affect the SIL results. The PC used for running DUNE is some magnitudes faster than the BBB, and so uses shorter time to compute, among other calculations, the optimization. These two differences are most prominent. The SIL results are still valuable, even though most variables, for instance controller tuning, has to be changed in a real-life flight test based on results from other flight tests. Some credibility can be accounted to SIL testing, as the results for LS path planner using the waypoint controller have similar results in SIL and in flight tests, see Section 7.3.

The largest error to the SIL tests may be the calculation of the optimization for OWSI and the Optimal path planners. As the optimization is calculated very often in the Optimal path planner, there is a chance that the calculation will use most of the computational power of the BBB. If the optimization was calculated at every received estimated state (10 Hz (Pixhawk Autopilot, 2014)), and the maximum calculation time is used (100 ms, found in Section 6.1.1) this would use all the BBB's CPU. Therefore, simulations of the Optimal approach with optimization on every estimated state will most likely not be accurate. As the simulations used for testing the path planner only performed an optimization every fifth estimated state, the BBB's CPU should be able to replicate the computational performance in real world flight tests also.

It is very probable that the z-direction velocity error measurement gathered in the second test of Section 5.1.3 was wrong. This value was always zero, which is not likely to be true.

The most valuable SIL results were not the achieved accuracy and precision itself, but which approach strategies and flight controllers that worked the best, as the errors mentioned above will probably not affect that result unless the computation time leads to missed deadlines.

To compare the waypoint controller to the LOS and SMC controller in SIL Test 1, 10 comparisons were run in SIL, and the LOS controller was found to be more accurate than the waypoint controller (Table 5.7 and 5.8). Also, the LOS controller was needed for the optimization-based approaches to work, so it had to be used anyway. The most probable reason that the LOS controller performed better, is that the waypoint controller is easily blown off the path as mentioned in Section 2.4.1.

The best approach strategy was found in SIL by comparing how close each method came to the CARP and the required velocity. From Table 5.9 to 5.11 and the results in Table 5.12, it is simple to see that the Optimal path planner combined with the LOS and SMC controller is the best on all accounts except velocity accuracy. The OWSI approach is a little better on that account with 0.5968 m/s compared to 0.6126 m/s. Flight tests should also be used to confirm these results.

Another point in favour of the Optimal and OWSI path planners is that the accuracy of the LS path planner in body frame y-position was much lower than for the Optimal and OWSI path planners. Errors in this direction are most likely caused by the LOS controller's path following errors. These errors are in theory somewhat mitigated when continuously creating new optimal paths like the Optimal and OWSI path planners does, as stated in Section 2.5.3.

One baffling result from the SIL tests is the terrible OWSI precision in y-direction. It ought to be pretty close to the Optimal path planner's precission, but it is not (7.1807 m for OWSI and 2.8600 m for Optimal). It might be the result of a bad simulation where the PC goes into a power saving mode.

A predicted result from the CARP detection in Section 2.6 is that the mean error in x-direction should be within the decision radius (1.7 m in simulations) when the mean error in yz-direction is low, as it is using Optimal and OWSI. This is due to the spherical detection close to the CARP. The CARP detection also predicts that the mean error in x-direction should be positive when mean error in yz-direction is high, as it is for the LS path planner. A recommendation for future simulations is to lower the decision radius.

Results from the wind estimator SIL testing are very important as it is only in a simulated environment wind can be set and known precisely. The two conducted tests have quite different results. This might be due to the attitude not being changed enough at the start of the 5 m/s wind test. The results of such low attitude change can be noticed when looking at the y wind error in Figure 5.3.

Trajectory simulations in MATLAB were made on the premiss that the wind was restricted to fluctuations about 180 ± 10 degrees, the absolute wind speed varying about 5 m/s ±50%. This creates a few uncertainties to the simulated wind forces. The initial speed and position are not known precisely in real world application, but are assumed to be so in these simulations. Also due to wind gusts and a non-uniform wind spectra in a real world application, the uncertainties of the impact point would probably increase. Another factor regarding winds, is that stronger winds and/or a downwards/upwards wind-speed components could occur in the harsh Arctic environment. However, the simulations give a good overview for how wind might affect the beacon during a free fall.

Release altitude was also a main contribution to the impact point uncertainty, as the uncertainty increases with the release altitude, see Figure 5.2.

Finally, the simulations are viewed as good pointers of wind and release altitude as a sources of uncertainty. This is the same conclusion as in Fuglaas (2014).

7.2 Discussion on Hardware Tests

Testing optimization on the BBB was performed to see how quickly it could do the calculation. The measurements were not recorded, but all were within the span indicated in the test section, Section 6.1.1. A computer running an OS always has sources of error as it is has parallel processes, some with higher priority than the optimization. Effects of variations in temperature and voltage are among many of the physical error sources applying to this test. As a high number of calculations were completed and timed in this test, and its goal was to find the worst case scenario, its results are assumed to be correct.

A test was performed to see whether the release mechanism worked or not. During the test, described in Section 6.1.2, there were several possible results. As the PWM generator had in part been tested before in Section 3.3.4, it was known that it could indeed produce a PWM signal. The possible outcomes for the test would then be either:

- No effect on drop mechanism, which could mean an error anywhere in the system
- Drop mechanism working continuously, would most probably mean that a low signal was sent from the BBB at all times
- Drop mechanism opening and not closing, which would most probably mean a failure in the PWM generator, where it would send a PWM signal at all times
- Drop mechanism working for five seconds and stopping for two seconds, which would most probably mean an inversion in the signal from the BBB
- Drop mechanism working for two seconds and stopping for five seconds, the only correct

result

As the test was executed many times in a row, a random event would most likely not be the reason for the reaction. The test continuously produced the last outcome. The voltage on the signal from the BBB to the PWM generator was also measured during testing for verification of an alternating high/low signal.

The second hardware test was performed to confirm that the BBB could communicate with the Pixhawk, and simultaneously check that the IMC bus was working across the radio link. The test would also show whether the ground station software was working as intended.

This test could experience a lot of different failures. Examples are

- Loss of radio link
- IMC messages not sendt or received
- Failure of communication between the BBB and the Pixhawk
- Internal errors in any component

The possible outcomes could be split into either Neptus not receiving autopilot data, which would mean a failure, or Neptus receiving sensible autopilot data, which would mean success.

Neptus received data and the test proved a success. Again, the test was executed continuously so it is not probable that it happened at random. The Pixhawk was also moved around to confirm that the data received on Neptus was sensible.

In this parts' final test, the full UAV system was tested. As it transmitted the correct data to the ground station, it was concluded that the system worked well.

7.3 Discussion on Flight Tests

As all the tests led to a measurement during a drop or a signal to drop, they were successful, with the exception of the test where the UAV would not start traversing the long stretch due to too high speeds. There was no need to actually drop the test beacons, as the target position was chosen using a map of Eggemoen in Neptus. This meant that we only knew with very low accuracy where we were aiming. There was also no way to record or measure where the test beacons landed other than watching the drop or trying to find scratches in the ice, which are quite inaccurate methods.

What was actually used to measure drop error, was the estimated position and its proximity to the CARP during the signalling of a drop. The estimated position is considered accurate to 1.3869 m and precise to and MRSE of 0.98.

The results achieved showed a difference in the accuracy of the approach strategies. During actual testing in gentle breeze and breeze, the ILS approach was more accurate than the LS approach. If the ILS approach is viewed as an approximation to the LS approach using a LOS controller, one could conclude that a LOS controller is more accurate than the waypoint controller as found in simulations.

The average accuracy of the LS path planner using the waypoint controller in flight tests is actually quite close to the same approach method in simulations, 17.47 m average in SIL compared to 14.52 m in flight tests, giving some credibility to all performed SIL tests.

As only three samples of each approach were collected, the amount of data makes the results inconclusive. It is also important to note that as this experiment was carried out early on in the project, the drop velocity error was not recorded, even though it is a large source of error.

Some other improvements and possibilities for improvements were found during the flight test. The X8 catapult was not as compatible with the drop mechanism position as first thought: the test beacons were placed at the point of the wings where the catapult was connected with the X8. This led to take-off being harder to achieve successfully. Another improvement was to use more rubber bands so that the catapult would be powerful enough when the X8 held a test beacon.

Other problems were that the drop mechanisms were activated when restarting DUNE and that the test beacons got stuck to the X8. Both were fixed by changing the drop mechanism placement and shape. DUNE restarting was fixed after the test was performed.

7.4 Theoretical System Accuracy

In this section, the system's theoretical accuracy will be discussed with a primary focus on the different sources of error.

7.4.1 Errors in Approach Methods

While flying, the UAV uses the controllers described in Section 2.4 and the path planners described in Section 2.5. These will always have some error due to disturbances. Tuning might make the errors smaller, but disturbances will always occur. Slow disturbances, such as wind, will disappear due to the sliding mode controller's course control. Quicker sporadic disturbances, like wind gusts, will still have some effect (Fortuna and Fossen).

In SIL simulation (Section 5.1.3, Table 5.12), the mean errors and MRSE was as low as 1.47 m and 2.95 m in position and 0.61 m/s and 1.00 m/s in velocity. These results are for the Optimal path planner using LOS and SMC controller, and their errors depend on tuning.

As the fall is about 3 seconds long depending on release height, the velocity estimation drop accuracy and precision becomes 0.61 m/s * 3 s = 1.83 m and 1.00 m/s * 3 s = 3 m when mapped onto the ground.

7.4.2 Drop Timing and CARP Calculation

When the CARP is calculated, it uses a step length for the iterator and a given air resistance constants (Section 4.4.3 and Section 3.1.9). Unless the step length is sufficiently small and the constants are given accurately, there will be errors in the trajectory calculations. But, as shown in Section 2.11, the step length used in SIL and on the BBB (0.001 s) should be very stable and be able to depict the trajectory realistically. Also, it has a zero detection accuracy of 0.042 m.

Deciding whether to drop the beacon or not is based on the CARP detection logic from Section 2.6. This has an additional error of 0.85 to 1.70 m depending on the decision radius. The error was about 1.7 m in SIL as predicted, and so if a shorter decision radius is used, the CARP detection accuracy should become better.

7.4.3 Sensory Input Timing and Accuracy

As the accuracy and timing of the drop depends on the sensor suite, the relevant data and conclusions will be stated in the below subsections.

GPS and IMU

The Pixhawk uses a differential GPS with a 5 hz update rate and 2.5 m accuracy in position (Section 3.1.7). Due to interaction with the IMU, much more accurate data is supplied from the state estimation (Section 7.4.3). The IMU consists of several sensors. Their names, types and accuracies are found at the Pixhawk web site: Pixhawk sensors.

State Estimation

The state estimation is based on the IMU and GPS data, where the IMU uses the GPS as a reference input to its Kalman filter to get as accurate estimates as possible. By doing this, measurement noise and bias are minimized, while update rate goes up. For the current Pixhawk settings, the update rate was 10 Hz (Section 3.1.7).

From the plots and measurements in Section 6.2.1, the estimation accuracy and precision is found to be 1.39 m and 0.98 m. The error in down direction is quite large, and is affecting the final measurement results a lot.

Wind Estimation

The inaccuracy of the wind estimate in Section 5.1.2 has more impact on the free fall trajectory than on the position error of the UAV. As the estimation accuracy is 0.78 m/s and has a precision of 0.45 m/s, the wind estimation drop accuracy and precision becomes 0.78 m/s * 3 s = 2.34 m and 0.45 m/s * 3 s = 1.35 m if the attitude change is large enough for the system to be sufficiently observable.

7.4.4 Optimization Time

The most demanding computational task that the payload computer performs is to calculate the FARP and choose the CARP. This was timed on the payload computer, and it used between 60 and 100 ms (Section 6.1.1). Linear computational time is assumed for all variables, O(n + h + t) using the same time step. Assuming worst case scenario, 100 ms delay at 17 m/s speed leads to 1.7 m error in x-direction. If it is also assumed that the optimization takes place for every fifth received estimated state at ten hertz, as in SIL Test 2, the mean delay error becomes 0.34 m, with a variance of 0.58 m.

7.4.5 OS Real Time Capabilities

GLUED is based on Linux, and so is not real time. However, GLUED running DUNE on the BBB seems to be very responsive and has very little lag before performing the release. Due to inherent lag in the MAVlink to IMC translation, the UAV lab normally works with a 20 ms response time. This adds up to 0.34 m error in x-direction.

7.4.6 Combined Theoretical Drop Accuracy

All the delays and errors from 7.4 are combined to make a realistic drop scenario. The errors listed below are based on the Optimal path planner with LOS and SMC controller, and a release height of 30 m in SIL. SIL measurements show that the fall time is about three seconds. Using this in the error calculation means that all velocity errors are multiplied by the fall time to deduce the ground impact point error, which is the same as error on hitting the target. All timed errors are multiplied by the UAV speed set point, 17 m/s, to deduce the ground impact point error. The axes are assumed uncorrelated and are added using the sum in quadrature, while calculating the precision using MRSE or DRMS. The zero detection error is a maximum error and so measured only in accuracy. CARP detection error is already included in the approach method error.

Some errors are assumed to only affect one direction: zero detection affects the z-accuracy, and the lags affect x-accuracy. Errors in z-direction are assumed to have the effect drawn in Figure 7.1, which according to Newton's second law with constant acceleration from gravity (g) and no air drag gives

$$s = vt + \frac{at^2}{2} \tag{7.1}$$

where s is distance, v is start speed, a is acceleration and t is time. Inserting with errors and finding the differences in x to the trajectory gives

$$\gamma = v_x(t - t_f) \tag{7.2}$$

$$t = \frac{-v_z \pm \sqrt{v_z^2 + 2gs_r}}{g} \tag{7.3}$$

$$t_f = \frac{-(v_z + v_{err}) \pm \sqrt{(v_z + v_{err})^2 + 2g(s_r - z_{err})}}{g} \tag{7.4}$$

where γ is the z-error to x-direction from Figure 7.1, t and t_f are the positive solutions to the fall time with and without errors, v_x and v_z is the speed in x and z-direction, s_r is the release height, v_{err} is the z-velocity error and z_{err} is the z-position error.

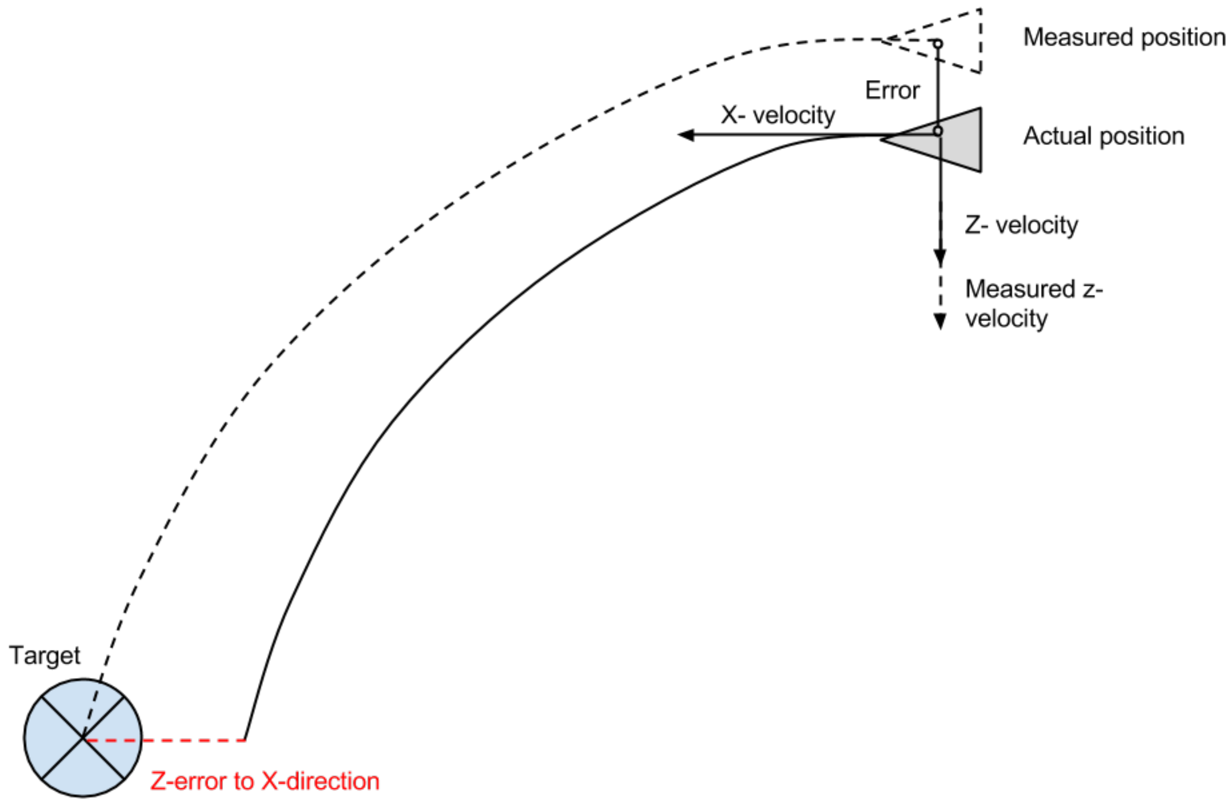


Figure 7.1: Error propagation in z-direction

Gathering the z errors gives us

Error Source	**Z-variance**	**Z-mean**
Approach method position	2.31 m	-0.46 m
Approach method velocity	0 m/s	-0.0 m/s
State estimate position	0.40 m	-1.36 m
Wind estimate	0 m/s	-0.32 m/s
Zero detection	-	-0.042 m

Table 7.1: Z error for ground impact point

Adding them and putting them in 7.1 to 7.4 gives the mapped x-error.

$$t = \frac{-0 \pm \sqrt{0^2 + 2 * 9.81 * 30}}{9.81} = 2.47[s] \tag{7.5}$$

$$t_f = \frac{-(0+0.32) \pm \sqrt{(0+0.32)^2 + 2 * 9.81 * (30 - 0.042 - 1.36 - 0.46)}}{9.81} = 2.36[s] \tag{7.6}$$

$$\gamma = 17(2.36 - 2.47) = -1.88[m] \tag{7.7}$$

$$\tag{7.8}$$

where the sign of the wind estimate is switched as the UAV experiences the opposite of the wind flow. Mapping the standard deviation (square root of variance) as well gives

$$t = \frac{-0 \pm \sqrt{0^2 + 2 * 9.81 * 30}}{9.81} = 2.4731[s] \tag{7.9}$$

$$t_f = \frac{-(0) \pm \sqrt{(0)^2 + 2 * 9.81 * (30 + \sqrt{2.31 + 0.40})}}{9.81} = 2.54[s] \tag{7.10}$$

$$\gamma = 17(2.54 - 2.47) = 1.14[m] \tag{7.11}$$

$$\tag{7.12}$$

Giving the variance 1.30 m.

Error Source	**X-mean**	**Y-mean**
Approach method position	-1.37 m	0.27 m
Approach method velocity	-0.13 m/s * 3 s = -0.39 m	0.60 m/s * 3 s = 1.8 m
State estimate position	-0.23 m	-0.10 m
Wind estimate	0.59 m/s * 3 s = 1.77 m	-0.39 m/s * 3 s = -1.17 m
Response time, BBB	0.34 m	-
Optimization lag	0.34 m	-
Z-error mapped	-1.88 m	-
Sum	-1.42 m	0.80 m

Table 7.2: Accuracy for ground impact point

Table 7.2 and 7.3 lead to the final drop accuracy and precision:

Error Source	**X-variance**	**Y-variance**
Approach method position	3.53 m	2.86 m
Approach method velocity	0.45 m/s * 3 s = 1.35 m	0.56 m/s * 3 s = 1.68 m
State estimate position	0.23 m	0.33 m
Wind estimate	0.11 m/s * 3 s = 0.33 m	0.09 m/s * 3 s = 0.27 m
Optimization lag	0.58 m	-
Z-error mapped	1.30 m	-
Sum	7.59 m	5.14 m

Table 7.3: Precision for ground impact point

Accuracy, 2. norm $\sqrt{1.42^2 + 0.80^2} = 1.63$ m

Precision, DRMS $\sqrt{7.59 + 5.14} = 3.57$ m

Precision, 2DRMS $2\sqrt{7.59 + 5.14} = 7.14$ m

where 2DRMS is the circle with a 95% chance to hit within (GPS Position Accuracy, 2003).

In addition to these sources of error, the actuator release time compensation might be off, the trajectory calculation using Euler's method has errors (estimating C_d, and step length, the beacon might bob or spin), and the state estimate velocity estimate error is unknown. The drop mechanism actuator release time is probably measured correctly down to ±0.1 s (* 17 m/s = ±1.7 m) (Section 3.1.2), while the trajectory errors are hard to determine without a large amount of tests. The velocity estimate error is also hard to determine, as no reference velocity is available, but it is probably in the same order as the position estimate.

7.5 Design Choices and System Comparison

As the thesis progressed, more and more advanced methods were used. The benchmark used in the start of the thesis was the Adapted Dubins Path from Fuglaas (2014). Using a variation of this, the LS path planner (Section 2.5.1), combined with the Pixhawk's waypoint controller (Section 2.4.2) led to varying results (Table 5.8). The results depended on how well the UAV entered the LS path and the disturbances affecting it along the path, such as wind gusts. Using the

ISL path planner (Section 2.5.2) compensated somewhat for the disturbances and gave better results (Section 6.2.3). As this was still unsatisfactory, a LOS controller (Section 2.4.1) was implemented and used on the LS approach, and the path following accuracy improved drastically (Table 5.7). This system worked very well, but could become inaccurate when the UAV wavered from the LS path. These incidents inspired the Optimal approach (Section 2.5.3). The results became even better (Table 5.9), but some downsides were found: the approach method could make the UAV start oscillating when the path changed too often, and the optimization calculations might take too long. To diminish these downsides, the OWSI approach (Section 2.5.4) was created. This approach optimized more seldom and removed the oscillations, but lost some accuracy.

To find the best approach method, all parts of the method must be considered. The different parts are the wind estimator, the path planner and the flight controllers.

From the controller test provided in Section 5.1.3, it is simple to see that the LOS with SMC gives better performance on path following than the unpredictable waypoint controller.

The wind estimators tested was the Pixhawk's built in open source wind estimator and the wind estimator from Johansen et al. (2015). Both worked quite well, but Pixhawk's wind estimator was more accurate, while Johansen's was more precise. Both have their merits, but Johansen's wind estimator was used for the most part during testing.

The path planners have all got their advantages as seen in Section 5.1. The LS path planner creates a simple path that is easy to follow, but has the lowest accuracy and precision. The ILS path planner is a better version of the LS path if the waypoint controller is used. Both of these assume that the wind remains the same throughout the drop procedure. The Optimal path planner makes more difficult paths that might cause the UAV to oscillate, but if the controllers are tuned correctly, the oscillations will be small or non-existent. The Optimal path always uses the newest wind estimate, and may provide better paths when the UAV is off course. The OWSI path planner is likely to lessen the amount of oscillations, but updates the optimal path more seldom. It also has lower precision than the Optimal and the LS path. When all of this is taken into account, the Optimal path is normally the best to use.

Another advantage of the Optimal and OWSI paths becomes apparent if the sea ice is moving along a path. The UAV may then follow that path while compensating for side wind and updating the target at every optimization. The LS path is based on going up against the wind, and so would not necessarily follow the ice path. This combined with the LS path's constant CARP means that the target will be the estimated sea ice centre at the time of the beacon release. This will lead to large errors if the release time is not estimated correctly or if the sea ice path changes.

Chapter 8

Conclusion and Further Work

This chapter will discuss the objectives of this thesis and sum up its results. It will also provide a summary of the work that has been done and inevitably, work that still remains.

8.1 Conclusion

The complete system was tested throughout the thesis using both simulated and flight tests, so that the best approach methods were found. It was quickly realized that the LOS and SMC controller provided better and more robust results than the waypoint controller, and through evolution of the approach methods, the OWSI path planner was created, even though the Optimal path gave better simulated results. Johansen's wind estimator proved to be more precise than the Pixhawk's wind estimator, and is therefore considered as the better choice.

The FARP was defined and used in the optimizations on which the Optimal and OWSI path planners are based. An analytical definition was not found for the FARP, and so a discrete optimization technique was used on a range of the FARP. The range was found by using an iterative solver.

The final result of a drop mission is whether or not the beacon will land and stay on the specified sea ice. The work in this thesis is based on being able to hit sea ice with a radius as small as 10 m. Whether or not this is possible is judged by the drop accuracy and drop precision.

The approach method using the Optimal path and LOS controller resulted in a theoretical drop accuracy of 1.63 m and a drop precision of 7.14 m measured in 2DRMS. Using the fact that 2DRMS gives the 95% hit chance radius, and assuming the accuracy to be in a circle around the target, this combines to a 95% chance of hitting within $7.14 + 1.63 = 8.77$ m of the target coordinate which is smaller than the target sea ice size set in this thesis.

This result was unfortunately not confirmed by flight tests, as unforeseen circumstances prevented flight time when the system was completed. However, the performed flight tests validates the correctness of the SIL environment to some degree.

8.2 Conclusion of Objectives

All objectives, except Objective 8, were completed, and the resulting system has been presented in this thesis.

Objective 1 was to implement and test a wind estimator. The implementation is given in Appendix H, the theory in Section 2.3 and testing in Section 5.1.2. The resulting accuracy was 0.78 m/s with an MRSE of 0.45 m/s.

Objective 2 was to calculate the FARP. The calculation is found in Section 2.1 and its implementation in Appendix H in the Beacon class code. An analytic solution was not found, but an iterative method was used instead.

Objective 3 was to design and test the LS path planner with SIL and flight tests. The design is given in Section 2.5.1, while testing is described in Section 5.1.3 and 6.2.2. Accuracy in SIL was 2.79 m in position and 0.74 m/s in velocity, while precision was 3.31 m in position and 1.23 m/s in velocity using the LOS controller.

Objective 4 was to optimize on the FARP and choose a CARP. This optimization is described in Section 2.5.3. Testing using this optimization in the Optimal and OWSI path planner are found in Section 5.1.3.

Objective 5 was to implement a LOS controller. Its implementation is found in Appendix H. The LOS controller was found to be superior to the waypoint controller in SIL Test 1 in Section 5.1.3.

Objective 6 was to research different approach methods to reach the CARP. They are described in Section 2.5 and tested in Chapter 5 and 6. The best accuracy achieved at reaching the CARP was 1.47 m in position and 0.61 m/s in velocity, while having a precision of 2.95 m in position and 1.01 m/s in velocity using the Optimal path planner with LOS and SMC controller.

Objective 7 was to SIL test the approach methods. This is found in Section 5.1.3. All results with accuracy and precision are found there.

Objective 8 was to field test the approach methods. This is described in Section 6.2. All results are found there. Unfortunately not all approach methods were tested in flight tests. The Optimal and OWSI path planners still has to be tested, as well as the LOS and SMC controllers and the wind estimator from Johansen et al. (2015). These flight tests were not performed due to several reasons stated in 8.3.

Objective 9 was to compare the LS path planner with waypoint controller to the best approach method found in the two objectives above. This is discussed in Section 7.5, where it was found that the Optimal path planner with LOS and SMC controller was the best as long as the controllers are well tuned.

8.3 Recommendations for Further Work

As in most projects, there is room for improvement. Several topics for further work were considered, both in short- and long term aspects.

Flight testing of the LOS with SMC controller, and OWSI and the Optimal path planner still remains. These tests were vital for the project results, but could not be performed in time due to several reasons. Change of staff delayed all flight tests in April. Bad weather prevented testing most of May, and in June hardware-in-the-loop (HIL) tests were required before autopilot flights could be flown, even though the HIL testing environment was not yet ready. So even though three months were set aside to be used for flight tests, most tests were not performed.

Remaining and recommended work is listed below.

Flight Tests Flight test for the Optimal and OWSI path planners, and for LOS with SMC controller must be performed to validate SIL results.

Investigating Z Velocity The SIL results show that velocity in z direction had no error. This is unlikely a true result, and the reason for this measurement should be investigated.

Statistically Viable Amount of Testing To make sure that the UAV's drop accuracy and precision were calculated and measured correctly, a large amount of drop tests is needed.

Error Mitigation Errors in accuracy and precision should always be minimized. Some of them might be mitigated, for example by using improvements like

- Magnetic release mechanism to remove drop time
- Avoid gliding state by moving the drop mechanism
- RTK-GPS would probably better the state estimate accuracy
- Lower computing time and lag by increasing computing power
- Using better controllers, faster actuators and/or using more acrobatic aircraft
- Higher frequency state estimation

Beacon Ground Impact Error Analysis and Mitigation As the beacon hits the ground, it will probably start rolling, skidding or jumping, further increasing error in beacon placement. This could be taken into account when calculating the CARP, or the beacon could be modified with spikes (or other measure) to stop it on impact.

Colder-Weather Testing As the UAV is supposed to work in Arctic conditions, it should be flown in such conditions. In this thesis, the coldest test was about -5 to -10 C.

Testing of Other LOS controllers Many other LOS controllers are available for fixed-wing UAVs. These may provide better accuracy on reaching the CARP, and therefore make a more accurate system. The Pixhawk's built in L1 controller seems to be well suited.

Combine With Other Modules This thesis describes one of the modules mentioned in Section 1.1. Putting it together with the other modules would complete the final system.

Moving Ice Prediction As the ice would move after its position has been estimated, it would be useful to know its path. Optimizing the UAV LOS path for the ice path would provide a better chance of the beacon landing on the ice.

Beyond Line-of-Sight Testing To complete a drop mission on ice as described in Section 1.1, the system must be able to fly beyond line of sight.

Appendix A

Acronyms

NTNU Norges Teknisk Naturvitenskapelige Universitet - Norwegian University of Science and Technology

LSTS Laboratório de Sistemas e Technologias Subaquáticas - Underwater Systems and Technology Laboratory

ITK Institutt for Teknisk Kybernetikk - Department of Engineering Cybernetics

DRMS Distance Root Mean Squared

MRSE Mean Radial Spherical Error

GPS Global Positioning System

RTK-GPS Real-time kinematic GPS

UAV Unmanned Aerial Vehicle

ISL Incremented-Straight-Line

SL Straight-Line

SMC Sliding Mode Controller

AUV Autonomous Underwater Vehicle

BBB BeagleBone Black

POE Power Over Ethernet

FDDI Fiber-Distrubuted-Data-Interface

PCB Printed Circuit Board

DUNE Unified Navigation Environment

FDDI Fiber Distributed Data Interface

RTTI Run-Time Type Information

IMC Inter-Module Communications

PWM Pulse Width Modulated

IDE Integrated Development Environment

LAN Local Area Network

LOS Line-of-Sight

DHCP Dynamic Host Configuration Protocol

IP Internet Protocol

SSID Service Set Identifier

SIMD Single Instructions, Multiple Data

HIL Hardware In the Loop

SIL/SITL Software In The Loop

IAS Indicated Air Speed

SD Secure Digital

BIOS	Basic Input-Output System
GPIO	General Purpose Input and Output
CAN	Controller Area Network
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
OWSI	Optimize When Success is Improbable

Appendix B

X8 Payload Checklist

The payload checklist shall be completed before every flight. The following tests shall be performed:

1. See to that all wires are connected correctly, except the batteries
2. Connect flight battery to the Pixhawk
3. Check the Pixhawks power light and look for error codes
4. Be sure that the Pixhawks speaker start up signal sounded (if speaker is in use)
5. Check the Rocket M5 power light
6. Check the router power supply
7. Connect the payload battery to the cape
8. Check the BeagleBone Blacks power light
9. Be sure that the ground station is running all programs
10. Send release test from ground station to X8 and check that release is executed
11. Get position from X8 sent to ground station

Appendix C

Flight Plans

Two flight plans were made in preparation to flight tests in the spring 2015. The tests shall be conducted on Agdenes Airfield. The flight plans are listed here.

C.1 Flight Plan: Following Waypoints

One test will be to follow waypoints to tune the controllers.

1. Take off using launch mechanism
2. Pilot flies to appropriate height
3. Autopilot is turned on and follows prepared waypoints
4. When finished, loiter at end point
5. Pilot takes control over UAV
6. Manually land on airstrip

C.2 Flight Plan: Drop Test

Another test will be to see if the beacon drop works, and if so, check how accurate it is.

1. Take off using launch mechanism
2. Pilot flies to appropriate height and turns on loiter mode
3. Target is sent to UAV and test starts
4. When finished, loiter at end point
5. Pilot takes control over UAV
6. Manually land on airstrip

Appendix D

Installing GLUED and Bootloader on the BeagleBone Black

This list describes the procedure for installing GLUED and bootloader on the BBB:

- Download GLUED from GLUED (2014)
- Run command "./mkconfig.bash list" to see available system files
- Make configuration file by running "./mkconfig.bash <system_file>"
- Build GLUED by running "./mksystem.bash <configuration_file>"
- Prepare GLUED to be transferred to memory device with "./pkrootfs.bash <configuration_file>".
- Transfer prepared files to memory device using "./mkdisk.bash <configuration_file> <memory_device>"
- Plug memory device into embedded device
- Boot embedded system from memory device

The system file used in bulletpoint three was ntnu_x8_006.

The last bulletpoint on this list is different on most embedded devices. The BBB needed to have its bootloader changed to be able to boot from the memory device used in this project. The memory device used was a microSD card. To change the bootloader, the following procedure was followed:

- Insert microSD into the BBB microSD card slot
- With charger unplugged, connect the BBB to the PC using a USB-TTL cable
- Establish connection with 115200, 8N1
- Press switch S2 (near USB connector) and hold while plugging 5V charger
- By now you should be able to access GLUED rootfs on your computer
- Run command: "mount /dev/mmcblk1p1 /mnt"
- Run command: "rm -rf /mnt/*"
- Run command: "cp /boot/MLO /boot/u-boot.img /mnt"
- Run command: "umount /mnt"
- Run command: "sync (more than one time might be required)"
- Run command: "halt"
- Unplug charger

Appendix E

Cross Compiling DUNE for the BeagleBone Black

The version of DUNE used in this project can be found by cloning *git@uavlab.itk.ntnu.no:uavlab/dune.git*, and then cloning *git@uavlab.itk.ntnu.no:uavlab/dune-ntnu.git* to the dune/user folder. The dune folder should use the temp/drop-test-may branch, while the dune/user folder should be in the feature/precisionAirDrop branch.

This list describes the procedure for cross compiling DUNE for the BBB:

- Toolchain was made by running "./pktoolchain.bash <configuration_file>" in the directory where GLUED was downloaded
- Running "cmake -DCROSS=/home/PATH_TO_GLUED/lctr-??xx/toolchain/bin/ARCH [PATH_TO_DUN to cross compile DUNE. It should be run in a folder where the build files are supposed to be, for instance an empty folder called "build". The path described after "-DCROSS=" should point to the compiler made in the GLUED toolchain
- Then a package must be made from the compiled DUNE files by running "make; make package;"
- Then the package must be transferred to the embedded device by using rsync: "rsync -avz dune-*-*tar.bz root@SYSTEM_IP:/opt/lsts/dune/"

- To log onto the embedded device, SSH can be used: "ssh root@SYSTEM_IP". Default password is "root"
- When using SSH, run: "services dune restart"
- To run DUNE, choose an .ini file and run: ./dune -c [PATH_TO_.INI_FILE]

Appendix F

A8 Description by Texas Instruments

The AM335x microprocessors, based on the ARM Cortex-A8 processor, are enhanced with image, graphics processing, peripherals and industrial interface options such as EtherCAT and PROFIBUS. The devices support high-level operating systems (HLOS). Linux® and Android™ are available free of charge from TI.

The AM335x microprocessor contain the subsystems shown in and a brief description of each follows:

The microprocessor unit (MPU) subsystem is based on the ARM Cortex-A8 processor and the PowerVR SGX Graphics Accelerator subsystem provides 3D graphics acceleration to support display and gaming effects.

The Programmable Real-Time Unit Subsystem and Industrial Communication Subsystem (PRU-ICSS) is separate from the ARM core, allowing independent operation and clocking for greater efficiency and flexibility. The PRU-ICSS enables additional peripheral interfaces and real-time protocols such as EtherCAT, PROFINET, EtherNet/IP, PROFIBUS, Ethernet Powerlink, Sercos, and others. Additionally, the programmable nature of the PRU-ICSS, along with its access to pins, events and all system-on-chip (SoC) resources, provides flexibility in implementing fast, real-time responses, specialized data handling operations, custom peripheral interfaces, and in offloading tasks from the other processor cores of SoC (AM3358, 2014).

Appendix G

Hardware Testing Code

The code used for testing is found in this appendix.

G.1

```
[Include ../../common/transports.ini]

[PrecisionAirdrop.vegardTest]
Enabled = Always
Entity Label = TESTING
```

G.2

```
//
*****************************************************************
*************
// Copyright 2007-2014 Universidade do Porto - Faculdade de
Engenharia      *
// LaboratÃ³rio de Sistemas e Tecnologia SubaquÃ¡tica (LSTS)
*
//
*****************************************************************
*************
// This file is part of DUNE: Unified Navigation Environment.
*
//
*
// Commercial Licence Usage
*
// Licencees holding valid commercial DUNE licences may use
this file in    *
// accordance with the commercial licence agreement provided
with the        *
// Software or, alternatively, in accordance with the terms
contained in a  *
// written agreement between you and Universidade do Porto.
For licensing   *
// terms, conditions, and further information contact
lsts@fe.up.pt.        *
//
*
// European Union Public Licence - EUPL v.1.1 Usage
*
// Alternatively, this file may be used under the terms of the
EUPL,        *
// Version 1.1 only (the "Licence"), appearing in the file
LICENCE.md        *
// included in the packaging of this file. You may not use
this work         *
// except in compliance with the Licence. Unless required by
applicable      *
// law or agreed to in writing, software distributed under the
Licence is    *
// distributed on an "AS IS" basis, WITHOUT WARRANTIES OR
CONDITIONS OF     *
// ANY KIND, either express or implied. See the Licence for
the specific    *
// language governing permissions and limitations at
*
// https://www.lsts.pt/dune/licence.
*
//
*****************************************************************
*************
// Author: Vegard Grindheim
*
//
*****************************************************************
*************
```

```
// DUNE headers.
#include <DUNE/DUNE.hpp>
#include <DUNE/Hardware/GPIO.hpp>
#include <boost/numeric/odeint.hpp>
#include <boost/bind.hpp>

namespace PrecisionAirdrop
{
  namespace vegardTest
  {
    using DUNE_NAMESPACES;

    struct Task: public DUNE::Tasks::Task
        {
          GPIO* m_out;
          //! Constructor.
          //! @param[in] name task name.
          //! @param[in] ctx context.
          Task(const std::string& name, Tasks::Context& ctx):
            DUNE::Tasks::Task(name, ctx),
        m_out(NULL)
      {

      }

      //! Update internal state with new parameter values.
      void
      onUpdateParameters(void)
      {
      }

      ~Task(void)
            {
              onResourceRelease();
            }

      //! Reserve entity identifiers.
      void
      onEntityReservation(void)
      {
      }

      //! Resolve entity names.
      void
      onEntityResolution(void)
      {
      }

      //! Acquire resources.
      void
      onResourceAcquisition(void)
      {
       m_out = new GPIO(2);
                m_out->setDirection(GPIO::GPIO_DIR_OUTPUT);
                m_out->setValue(0);
      }
      //! Initialize resources.
      void
      onResourceInitialization(void)
      {
      }

      //! Release resources.
      void
      onResourceRelease(void)
      {
       //Memory::clear(m_out);
      }
      void
       consume(const IMC::EstimatedState* e_state)
       {

       }
      //! Main loop.
      void
      onMain(void)
      {
        inf("kjorer!");
        while (!stopping())
        {
          m_out->setValue(1);
          Delay::waitMsec(5000);
                      m_out->setValue(0);
          Delay::waitMsec(2000);
          waitForMessages(1.0);
        }
      }
    };
  }
}

DUNE_TASK
```

G.3

```
// Test program running Ardupilot and sending data over radio
link

[Require ../../etc/uav/ardupilot.ini]

[General]
Vehicle                                         = x8-00

[Control.UAV.Ardupilot/Simulation]
Ardupilot Tracker                               = False

[Control.Path.Height]
Enabled                                         = Simulation

[Control.Path.Aerosonde]
Enabled                                         = Simulation

[PrecisionAirdrop.vegardTest]
Enabled = Always
Entity Label = pixhawkTest
```

G.4

```
//
*****************************************************************
*************
// Copyright 2007-2014 Universidade do Porto - Faculdade de
Engenharia      *
// LaboratÃ³rio de Sistemas e Tecnologia SubaquÃ¡tica (LSTS)
*
//
*****************************************************************
*************
// This file is part of DUNE: Unified Navigation Environment.
*
//
*
// Commercial Licence Usage
*
// Licencees holding valid commercial DUNE licences may use
this file in    *
// accordance with the commercial licence agreement provided
with the       *
// Software or, alternatively, in accordance with the terms
contained in a  *
// written agreement between you and Universidade do Porto.
For licensing   *
// terms, conditions, and further information contact
lsts@fe.up.pt.        *
//
*
// European Union Public Licence - EUPL v.1.1 Usage
*
// Alternatively, this file may be used under the terms of the
EUPL,        *
// Version 1.1 only (the "Licence"), appearing in the file
LICENCE.md        *
// included in the packaging of this file. You may not use
this work         *
// except in compliance with the Licence. Unless required by
applicable      *
// law or agreed to in writing, software distributed under the
Licence is    *
// distributed on an "AS IS" basis, WITHOUT WARRANTIES OR
CONDITIONS OF      *
// ANY KIND, either express or implied. See the Licence for
the specific    *
// language governing permissions and limitations at
*
// https://www.lsts.pt/dune/licence.
*
//
*****************************************************************
*************
// Author: Vegard Grindheim
*
//
*****************************************************************
*************
```

```
// DUNE headers.
#include <DUNE/DUNE.hpp>
#include <DUNE/Hardware/GPIO.hpp>
#include <boost/numeric/odeint.hpp>
#include <boost/bind.hpp>

namespace PrecisionAirdrop
{
  namespace vegardTest
  {
    using DUNE_NAMESPACES;

    struct Task: public DUNE::Tasks::Task
        {
          GPIO* m_out;
          //! Constructor.
          //! @param[in] name task name.
          //! @param[in] ctx context.
          Task(const std::string& name, Tasks::Context& ctx):
            DUNE::Tasks::Task(name, ctx),
        m_out(NULL)
      {
        //bind<IMC::SetLedBrightness>(this);
             bind<IMC::EstimatedState>(this);
      }

      //! Update internal state with new parameter values.
      void
      onUpdateParameters(void)
      {
      }

      ~Task(void)
            {
              onResourceRelease();
            }

      //! Reserve entity identifiers.
      void
      onEntityReservation(void)
      {
      }

      //! Resolve entity names.
      void
      onEntityResolution(void)
      {
      }

      //! Acquire resources.
      void
      onResourceAcquisition(void)
      {
       //m_out = new GPIO(2);
             //m_out->setDirection(GPIO::GPIO_DIR_OUTPUT);
             //m_out->setValue(0);
      }
      //! Initialize resources.
      void
      onResourceInitialization(void)
      {
      }

      //! Release resources.
      void
      onResourceRelease(void)
      {
       //Memory::clear(m_out);
      }
      void
       consume(const IMC::EstimatedState* e_state)
       {
         inf("Estimated state phi is %f", e_state->phi);
         //dispatch(IMC::EstimatedState* e_state, e_state->
phi);
       }
      //! Main loop.
      void
      onMain(void)
      {
        inf("kjorer!");
        while (!stopping())
        {
          //m_out->setValue(1);
          //Delay::waitMsec(5000);
             //m_out->setValue(0);
          //Delay::waitMsec(2000);
          waitForMessages(1.0);
        }
      }
    };
  }
}

DUNE_TASK
```

Appendix H

DUNE Code

The final code that was written for the UAV system is found in this appendix. The .ini file is just an example, and should be changed to contain the suitable variables and values for the specific drop mission.

H.1 .ini file

H.2 Code for the Drop-on-Target task

H.3 Code for the Actuator-Output task

H.4 Code for the Wind-Estimator task

H.5 Code for the SMC task

H.6 Code for the Beacon class header

H.7 Code for the Enum header

H.8 Code for the PWM generator

H.1

```
[Require ../../etc/uav/ardupilot.ini]

[General]
Vehicle                                    = ntnu-x8-006

[Supervisors.UAV.LostComms]
Enabled                                    = Never

[Navigation.WindEstimator]
Enabled                                    = Always
Entity Label                               = Wind Estimator
Sample Window Size                         = 1000
Q Size                                     = 1000
Trusted Limit                              = 1e-3

[Actuators.LogicOutput.BBB]
Enabled                                    = Hardware
Entity Label                               = BBB Out
Initial Output                             = 0
Pin                                        = 2


[Control.UAV.Ardupilot/AP-SIL]
Ardupilot Tracker                         = True
MotorStopTimeout                          = 7.0
RC 1 PWM MIN                              = 1000
RC 1 PWM MAX                              = 2000
RC 1 MAX                                  = 65.0
RC 2 PWM MIN                              = 1000
RC 2 PWM MAX                              = 2000
RC 2 MAX                                  = 2.0
RC 2 REV                                  = True
RC 3 PWM MIN                              = 1000
RC 3 PWM MAX                              = 2000
RC 3 MIN                                  = 10.0
RC 3 MAX                                  = 30.0

[Control.UAV.Ardupilot/Hardware]
Ardupilot Tracker                         = True
MotorStopTimeout                           = 6.0
RC 1 PWM MIN                              = 899
RC 1 PWM MAX                              = 2101
RC 1 MAX                                  = 45.0
RC 2 PWM MIN                              = 899
RC 2 PWM MAX                              = 2064
RC 2 MAX                                  = 2.0
RC 2 REV                                  = False
RC 3 PWM MIN                              = 954
RC 3 PWM MAX                              = 1901
RC 3 MIN                                  = 12.0
RC 3 MAX                                  = 22.0
Enabled                                   = Hardware
Entity Label                              = Autopilot
```

```
TCP - Address                           = 192.168.1.10
TCP - Port                              = 9801

[General]
Time Of Arrival Factor                   = 5.0


[Autonomy.DropOnTarget]
Enabled                                  = Always
Entity Label                             = Drop on Target
Altitude Interval                        = 1
Connection Timeout                       = 3
Distance To LSSP                         = 100
Start Point Distance From CARP           = 200
End Point Distance From CARP             = 400
Percent Accurate                         = 7
Radius                                   = 100
Release Height                           = 50
Speed                                    = 17
Drop Error                               = 50
Guidance Mode                            = LONG_STRETCH
Increments                               = 2
Safe Height                              = 100
Max Drop Current                         = 3
Glide Time                               = 2
Drop Time                                = 0.6
Use Bank Controller                      = true
Use Wind Estimator                       = true
Step Size                                = 0.001
Optimize Points                          = 30
Optimize Rads                            = 2
Direct To Optimal                        = false
Max Counter                              = 100000
Optimize Once                            = false
Optimation Rate Inverse                  = 10
Position Weight                          = 15
Velocity Weight                          = 1
Repeated Testing                         = true
#Course Error Const                      = 1
#Wind Difference Const                   = 1
#Path Error Horizontal Const                 = 1
#Path Error Vertical Const               = 1
#Speed Error Constant                    = 1
OWSI Min Distance                        = 20

[Control.Path.Height]
Enabled                              = Always
Height bandwidth                     = 40
Vertical Rate maximum gain           = 0.20
EstimatedState Filter                = ntnu-x8-006:Autopilot
DesiredZ Filter                      = ntnu-x8-006:Path Control

[Control.Path.SMC]
```

```
Enabled                                 = Always
Cross-track -- Monitor                  = false
Entity Label                            = Path Control


[Transports.UDP]
Filtered Entities                       = CpuUsage:Daemon,
                                            EstimatedStreamVelocity:Wind
Estimator

[Transports.Logging]
Transports                              = Acceleration,
                                          AngularVelocity,
                                          Announce,
                                          AutopilotMode,
                                          Brake,
                                          ControlLoops,
                                          CpuUsage,
                                          Current,
                                          DesiredHeading,
                                          DesiredPath,
                                          DesiredRoll,
                                          DesiredSpeed,
                                          DesiredVerticalRate,
                                          DesiredZ,
                                          EntityList,
                                          EntityState,
                                          EstimatedState,
                                          EstimatedStreamVelocity,
                                          FollowReference,
                                          FollowRefState,
                                          FuelLevel,
                                          GpsFix,
                                          IndicatedSpeed,
                                          LeaderState,
                                          LinkLevel,
                                          LogBookEntry,
                                          MagneticField,
                                          ManeuverControlState,
                                          PathControlState,
                                          PlanControl,
                                          PlanSpecification,
                                          PlanControlState,
                                          PlanDB,
                                          PowerChannelControl,
                                          Pressure,
                                          Reference,
                                          Rpm,
                                          RSSI,
                                          SetControlSurfaceDeflection,
                                          SetThrusterActuation,
                                          SimulatedState,
                                          StopManeuver,
            StorageUsage,
            Target,
            Temperature,
            TrueSpeed,
            VehicleCommand,
            VehicleMedium,
            VehicleState,
            Voltage
```

H.2

```
// Copyright 2007-2014 Universidade do Porto - Faculdade de Engenharia      *

// DUNE headers.
#include "Beacon.hpp"
#include "Enums.hpp"

// DUNE headers.
#include <DUNE/DUNE.hpp>


namespace Autonomy
{
  namespace DropOnTarget
  {
    using DUNE_NAMESPACES;
    using namespace std;

    struct Arguments
    {
      double accepted_distance_to_start_point;
      fp64_t start_point_distance_from_carp;
      fp64_t end_point_distance_from_carp;
      fp64_t radius;
      double speed;
      fp32_t max_current;
      fp32_t glide_time;
      fp32_t drop_time;
      fp64_t drop_error;
      fp64_t percent_accurate;
      fp32_t release_height;
      uint16_t altitude_accuracy;
      uint16_t connection_timeout;
      uint16_t increments_input;
      std::string guidance_mode_input;
      uint16_t safe_height;
      bool bank;
      int counter_max;
      double dt;
      bool use_wind_est;
      double opt_circle;
      int opt_points;
      bool direct_to_opt;
      bool optimize_once;
      int opt_rate_inverse;
      double w_pos;
      double w_vel;
      bool repeated_testing;
      double OWSI_min_distance;
      double accepted_path_const_horizontal;
      double accepted_path_const_vertical;
      double accepted_speed_const;
      double accepted_course_const;
      double accepted_wind_difference;
```

```
};

struct Task: public DUNE::Tasks::Task
{
  //! Constructor.
  //! @param[in] name task name.
  //! @param[in] ctx context.

  //Variables

  //True if initiation is done
  bool m_initiated;
  //True if wind measurement message is received
  bool m_wind_received;
  //True if target message is received
  bool m_target_received;
  //True while a follow reference plan is running
  bool m_follow_reference_is_running;
  //True while vehicle status is blocked
  bool m_isBlocked;
  //True while vehicle status is ready
  bool m_isReady;
  //True while motor is off
  bool m_isGliding;
  //True pre takeoff
  bool m_ground;
  //True until new estimated state message is processed
  bool m_new_EstimatedState;
  //True until new follow reference state message is processed
  bool m_new_FollowRefState;
  //True when long stretch end point is ready
  bool m_hasPoint;
  //Wind unit vector size in x and y
  double m_wind_unit_x, m_wind_unit_y;
  //The current current running through the motor
  double m_current;
  //Cross track error
  double m_ct_error;
  //LOS course error
  double m_course_error;
  //Height used for prev CARP optimatzion (OWSI)
  double m_opt_height;
  //Speed used for prev CARP optimatzion (OWSI)
  double m_opt_speed;
  //How many increments remain in guidance mode incremented long stretch
  int m_increments;
  //Wind estimation entity
  int m_wind_est_ent;
  //Wind estimation entity from ardupilot
  int m_ardupilot_wind_ent;
  //Fail or success when plan is finished
  uint8_t m_stop_type;
  //Previously measured distance to a point (CARP)
```

```
      fp64_t m_previous_distance;
      //Smallest distance measured to CARP
      fp64_t m_smallest_distance;
      //Time when glide started
      uint64_t m_t_start_glide;
      //Timer for drop reaction time
      uint64_t m_drop_reaction_time;
      //The GPS beacon to be dropped
      Beacon m_beacon;
      //The current state of the state machine
      States m_current_state;
      //The chosen guidance mode
      Modes m_guidance_mode;
      //Timer for gliding
      Clock timer;
      //End point
      Point m_end_point;
      //Start point
      Point m_start_point;
      //Counter for optimation rate
      int counter;
      //Weighing matrix for velocity
      Matrix m_W_vel;

      //Arguments from .ini-file
      Arguments m_args;

      //IMC messages
      IMC::Reference m_ref[LAST_STATE];
      IMC::PowerChannelControl m_pcc;
      IMC::Target m_target;
      IMC::EstimatedState m_estate;
      IMC::EstimatedStreamVelocity m_ewind;
      IMC::EstimatedStreamVelocity m_opt_wind;
      IMC::EstimatedStreamVelocity m_carp_ewind;
      IMC::FollowRefState m_follow_ref;
      IMC::DesiredSpeed m_dspeed;
      IMC::DesiredZ m_dz;
      IMC::Brake m_break;

      Task(const std::string& name, Tasks::Context& ctx):
        DUNE::Tasks::Task(name, ctx)
      {
        //Load parameters from .ini-file
        param("Altitude Interval", m_args.altitude_accuracy)
        .defaultValue("5")
        .units(Units::Meter)
        .description("Altitude following accuracy");

        param("Connection Timeout", m_args.connection_timeout)
        .defaultValue("10")
        .units(Units::Meter)
        .description("Response time before failure of followReference");
        param("Course Error Const", m_args.accepted_course_const)
        .defaultValue(".01")
        .description("Accepted course error / distance");

        param("Wind Difference Const", m_args.accepted_wind_difference)
        .defaultValue(".1")
        .description("Accepted wind error");

        param("Path Error Horizontal Const",
m_args.accepted_path_const_horizontal)
        .defaultValue(".2")
        .units(Units::Meter)
        .description("Accepted horizontal path error / distance");

        param("Path Error Vertical Const",
m_args.accepted_path_const_vertical)
        .defaultValue(".1")
        .units(Units::Meter)
        .description("Accepted vertical path error / distance");

        param("Speed Error Constant", m_args.accepted_speed_const)
        .defaultValue(".1")
        .units(Units::Meter)
        .description("Accepted speed error / distance");

        param("OWSI Min Distance", m_args.OWSI_min_distance)
        .defaultValue("5")
        .units(Units::Meter)
        .description("Closest distance to CARP for optimation with OWSI");

        param("Direct To Optimal", m_args.direct_to_opt)
        .defaultValue("10")
        .units(Units::Meter)
        .description("True to go directly to target");

        param("Max Drop Current", m_args.max_current)
        .defaultValue("1")
        .description("Maximum motor current allowed for drop");

        param("Glide Time", m_args.glide_time)
        .defaultValue("1")
        .description("Glide time needed to stop the motor");

        param("Drop Time", m_args.drop_time)
        .defaultValue(".6")
        .description("Glide time needed to stop the motor");

        param("Step Size", m_args.dt)
        .defaultValue(".01")
        .description("Step size for CARP calculation");

        param("Max Counter", m_args.counter_max)
        .defaultValue("10000")
        .description("Max counter for simulation");

        param("Distance To LSSP", m_args.accepted_distance_to_start_point)
        .defaultValue("15")
        .units(Units::Meter)
        .description("Acceptance circle around long stretch start point");

        param("End Point Distance From CARP",
m_args.end_point_distance_from_carp)
        .defaultValue("100")
        .units(Units::Meter)
        .description("Distance from CARP to LSEP");

        param("Start Point Distance From CARP",
m_args.start_point_distance_from_carp)
        .defaultValue("100")
        .units(Units::Meter)
        .description("Distance from CARP to LSSP");

        param("Percent Accurate", m_args.percent_accurate)
        .defaultValue("7")
        .units(Units::Meter)
        .description("Percent Accurate turning");

        param("Optimize Rads", m_args.opt_circle)
        .defaultValue("3.1416")
        .description("How many radians to use during optimation");

        param("Optimize Points", m_args.opt_points)
        .defaultValue("10")
        .description("How many points to check during optimation");

        param("Radius", m_args.radius)
        .defaultValue("100")
        .units(Units::Meter)
        .description("Radius of the turns");

        param("Release Height", m_args.release_height)
        .defaultValue("50")
        .units(Units::Meter)
        .description("Drop release height");

        param("Speed", m_args.speed)
        .defaultValue("17")
        .units(Units::Meter)
        .description("Speed during mission");

        param("Drop Error", m_args.drop_error)
        .defaultValue("20")
        .units(Units::Meter)
        .description("Drop within circle of this radius");
        param("Safe Height", m_args.safe_height)
        .defaultValue("100")
        .units(Units::Meter)
        .description("High enough to be safe");

        param("Increments", m_args.increments_input)
        .defaultValue("4")
        .description("Increments for INCREMENTED_LONG_STRETCH");

        param("Guidance Mode", m_args.guidance_mode_input)
        .values("INCREMENTED_LONG_STRETCH, OPTIMAL_LONG_STRETCH, LONG_STRETCH,
OWSI")
        .description("Guidance mode for mission");

        param("Use Bank Controller", m_args.bank)
        .defaultValue("false")
        .description("Controller mode for mission");

        param("Use Wind Estimator", m_args.use_wind_est)
        .defaultValue("false")
        .description("Use new wind estimator");

        param("Optimize Once", m_args.optimize_once)
        .defaultValue("true")
        .description("Optimize once (true) or continously (false)");

        param("Repeated Testing", m_args.repeated_testing) //Write into report
        .defaultValue("false")
        .description("Test repeatedly on same target");


        param("Optimation Rate Inverse", m_args.opt_rate_inverse)
        .defaultValue("1")
        .description("Inverse of the optimation rate");

        param("Position Weight", m_args.w_pos)
        .defaultValue("10")
        .description("Weighing of position during optimation");

        param("Velocity Weight", m_args.w_vel)
        .defaultValue("10")
        .description("Weighing of velocity during optimation");


        //IMC messages to receive
        bind<Target>(this);
        bind<EstimatedStreamVelocity>(this);
        bind<EstimatedState>(this);
        bind<FollowRefState>(this);
        bind<PlanControlState>(this);
        bind<PlanControl>(this);
        bind<VehicleMedium>(this);
        bind<Current>(this);
```

```
          bind<EntityList>(this);
          bind<PathControlState>(this);
        }

        //! Update internal state with new parameter values.
        void
        onUpdateParameters(void)
        {
          //Initiate variables
          m_initiated = false;
          m_target_received = false;
          m_wind_received = false;
          m_follow_reference_is_running = false;
          m_isBlocked = false;
          m_isReady = false;
          m_ground = true;
          m_new_EstimatedState = false;
          m_new_FollowRefState = false;
          m_hasPoint = false;
          m_isGliding = false;
          m_current_state = IDLE;
          m_beacon = Beacon();
          m_increments = m_args.increments_input;
          m_previous_distance = INT_MAX; //A lot.
          m_smallest_distance = INT_MAX; //A lot
          m_current = 250;
          m_stop_type = IMC::PlanControl::PC_REQUEST;
          m_pcc.name = "drop";
          m_wind_est_ent = -1;
          counter = 0;
          m_ct_error = 0;
          m_opt_height = m_args.release_height;
          m_opt_speed = m_args.speed;

          m_guidance_mode = guidance_mode_string_to_enum
  ((char*)m_args.guidance_mode_input.c_str());

          double W_vel[] = {m_args.w_vel, m_args.w_vel, m_args.w_vel};
          m_W_vel = Matrix(W_vel, 1, 3);
        }

        //! Reserve entity identifiers.
        void
        onEntityReservation(void)
        {
        }

        //! Resolve entity names.
        void
        onEntityResolution(void)
        {
        }
```

```
      //! Acquire resources.
      void
      onResourceAcquisition(void)
      {

      }

      //! Initialize resources.
      void
      onResourceInitialization(void)
      {

      }

      //! Releance:se resources.
      void
      onResourceRelease(void)
      {
      }

      //Stopping motor and starting glide
      void
      startGlide(){
        m_break.op=IMC::Brake::OP_START;
        dispatch(m_break);
        m_isGliding = true;
        inf("GLIDING");
        m_previous_distance = INT_MAX;
        m_t_start_glide = timer.getMsec();
      }

      //Starting glide mode
      void
      goToGlideMode(){
        m_t_start_glide = timer.getMsec();
        m_hasPoint = false;
        //IMPLEMENT LATER: m_stop_type = IMC::PlanControl::PC_SUCCESS;
        m_current_state = GLIDE_MODE;
        m_previous_distance = INT_MAX;
      }

      //Guidance strategy consisting of going to a start point and then turn
against the wind towards the CARP
      void
      guidanceLongStretch(void)
      {
        if(!m_hasPoint)
        {
          m_hasPoint = true;
          makePointFromCarp(GOING_TO_DROP_POINT, true, false, m_wind_unit_x,
m_wind_unit_y, 1, m_target.z + m_args.release_height);
        }
        if(m_new_EstimatedState)
        {
          m_new_EstimatedState = false;
          double lat, lon, height, distance_to_point = INT_MAX;
          getCurrentLatLonHeight(&lat, &lon, &height);

          if(!m_isGliding)
          {
            distance_to_point = distanceByTime(lat, lon, height,
m_args.glide_time + m_args.drop_time);

            if(isCloseToPoint(distance_to_point))
            {
              startGlide();
            }
            if(!m_isGliding){
              m_previous_distance = distance_to_point;
            }
          }

          else
          {
            distance_to_point = distanceByTime(lat, lon, height,
m_args.drop_time);

            if(isCloseToPoint(distance_to_point) && motorShutDownTimeHasRun())
            {
              drop(distance_to_point);
              goToGlideMode();
            }
            else
            {
              if(distance_to_point < m_smallest_distance)
              {
                m_smallest_distance = distance_to_point;
              }

              m_previous_distance = distance_to_point;
            }
          }
          checkForFailure();
        }
      }

      void
      updateOpt(double height)
      {
        m_opt_wind = m_ewind;
        m_hasPoint = true;
        double dt;
        uint64_t start = timer.getMsec();
        m_beacon.optimal_CARP(height, m_ewind, m_estate, m_args.dt,
                              m_args.counter_max, m_args.opt_circle,
m_args.opt_points, m_W_vel, m_args.w_pos, m_args.glide_time);
        dt = timer.getMsec() - start;
        // USE ON BEAGLEBONE! war("TIME FOR OPT: %f sec", dt);

        double vx_unit = m_beacon.get_CARP().vx / sqrt(m_beacon.get_CARP().vx
* m_beacon.get_CARP().vx + m_beacon.get_CARP().vy * m_beacon.get_CARP().vy);
        double vy_unit = m_beacon.get_CARP().vy / sqrt(m_beacon.get_CARP().vx
* m_beacon.get_CARP().vx + m_beacon.get_CARP().vy * m_beacon.get_CARP().vy);

        makeStraightLineOverCarp(vx_unit,  vy_unit);
      }

      //Guidance strategy consisting of going to a start point and then turn
against the wind towards the CARP but using incremented waypoints to get there
      void
      guidanceIncrementedLongStretch(void)
      {
        if(!m_hasPoint)
        {
          m_hasPoint = true;
          makePointFromCarp(GOING_TO_DROP_POINT, true, false, -m_wind_unit_x,
m_wind_unit_y, 1, m_target.z + m_args.release_height);
          dispatch_reference();
        }

        if(m_new_EstimatedState)
        {
          //Init vars and distances
          double lat = m_estate.lat, lon = m_estate.lon, height =
m_estate.height, distance_to_point = INT_MAX, distance_to_carp = INT_MAX,
distance_to_end = INT_MAX;
          WGS84::displace(m_estate.x, m_estate.y, m_estate.height, &lat, &lon,
&height);
          distance_to_carp = WGS84::distance(m_beacon.get_CARP().lat,
m_beacon.get_CARP().lon, 0, lat,  lon, 0);
          distance_to_point = WGS84::distance(m_ref[GOING_TO_DROP_POINT].lat,
m_ref[GOING_TO_DROP_POINT].lon, 0, lat,  lon, 0);
          distance_to_end = WGS84::distance(m_end_point.lat, m_end_point.lon,
0, lat,  lon, 0);

          if(distance_to_point < m_args.start_point_distance_from_carp/
m_args.increments_input ) //TEST AND PUT IN INI FILE
            m_hasPoint = false;

          if(distance_to_carp < m_smallest_distance)
          {
            m_smallest_distance = distance_to_carp;
          }

          //check for carp prox
          if(((distance_to_carp < m_args.drop_error) && distance_to_carp >
m_previous_distance) || distance_to_carp < 1)
          {
            //IMPLEMENT NEW DROP LATER
            oldDrop();
            m_hasPoint = false;
            //m_stop_type = IMC::PlanControl::PC_SUCCESS;
            m_current_state = GOING_TO_SAFE_HEIGHT;
          }

          //Check if near end point of long stretch
          if(distance_to_end < m_args.end_point_distance_from_carp/5)
          {
            war("FAILURE: CARP missed! Was as close as: %f",
m_smallest_distance);

            m_smallest_distance = INT_MAX;
            m_previous_distance = INT_MAX;

            m_hasPoint = false;
            //m_stop_type = IMC::PlanControl::PC_FAILURE;
            m_current_state = GOING_TO_SAFE_HEIGHT;
          }
          m_previous_distance = distance_to_carp;
        }
      }

      //Guidance strategy consisting of repeatedly optimizing the CARP and the
straight line above it
      void
      guidanceOptLongStretch(void)
      {
        if(m_new_EstimatedState)
        {
          m_new_EstimatedState = false;
          double lat, lon, height, distance_to_point = INT_MAX;
          getCurrentLatLonHeight(&lat, &lon, &height);

          if(!m_isGliding)
          {
            if((!m_hasPoint || !m_args.optimize_once) && (counter%
m_args.opt_rate_inverse == 0))
            {
              updateOpt(height);
            }

            distance_to_point = distanceByTime(lat, lon, height,
m_args.glide_time + m_args.drop_time);

            if((distance_to_point < m_args.drop_error && distance_to_point >
m_previous_distance) || distance_to_point < 1)
            {
              startGlide();
            }
            else
            {
              m_previous_distance = distance_to_point;
            }
          }
          else
          {
            distance_to_point = distanceByTime(lat, lon, height,
m_args.drop_time);

            if(isCloseToPoint(distance_to_point) && motorShutDownTimeHasRun())
            {
              drop(distance_to_point);
              goToGlideMode();
            }
            m_previous_distance = distance_to_point;
          }

          if(distance_to_point < m_smallest_distance && m_isGliding)
          {
            m_smallest_distance = distance_to_point;
          }
        }
        counter++;
        checkForFailure();
      }

      //To check whether success is improbable
      bool
      OWSI_update(double distance, double height, double speed){
        return (distance > m_args.OWSI_min_distance) && (sqrt(abs((m_ewind.x -
m_opt_wind.x) * (m_ewind.x - m_opt_wind.x) + (m_ewind.y - m_opt_wind.y) *
(m_ewind.y - m_opt_wind.y) + (m_ewind.z - m_opt_wind.z) * (m_ewind.z -
m_opt_wind.z))) > m_args.accepted_wind_difference
            || (abs(m_ct_error) / distance >
m_args.accepted_path_const_horizontal)
            || (abs(height - m_opt_height) / distance >
m_args.accepted_path_const_vertical)
            || (abs(speed - m_opt_speed) / distance >
m_args.accepted_speed_const)
            || (abs(m_course_error) / distance >
m_args.accepted_course_const));//ENDRE DETTE I RAPPORTEN
      }

      //Guidance strategy consisting of optimizing the CARP and the straight
line above it if success is improbable
      void
      guidanceOWSI(void)
      {
        if(m_new_EstimatedState)
        {
          m_new_EstimatedState = false;
          double lat, lon, height, distance_to_point = INT_MAX, speed = sqrt
(m_estate.vy * m_estate.vy + m_estate.vx * m_estate.vx);
          getCurrentLatLonHeight(&lat, &lon, &height);
          if(!m_isGliding)
          {
            if(OWSI_update(WGS84::distance(m_beacon.get_CARP().lat,
m_beacon.get_CARP().lon, 0, lat,  lon, 0), height, speed))
            {
              updateOpt(height);
              m_opt_height = m_beacon.get_CARP().z;
              m_opt_speed = speed;
            }

            distance_to_point = distanceByTime(lat, lon, height,
m_args.glide_time + m_args.drop_time);

            if((distance_to_point < m_args.drop_error && distance_to_point >
m_previous_distance) || distance_to_point < 1)
            {
              startGlide();
            }
            else
            {
              m_previous_distance = distance_to_point;
            }
          }
          else
          {
            distance_to_point = distanceByTime(lat, lon, height,
m_args.drop_time);

            if(isCloseToPoint(distance_to_point) && motorShutDownTimeHasRun())
            {
              drop(distance_to_point);
              goToGlideMode();
            }
            m_previous_distance = distance_to_point;
          }

          if(distance_to_point < m_smallest_distance && m_isGliding)
          {
            m_smallest_distance = distance_to_point;
          }
        }
        counter++;
        checkForFailure();
      }

      //Guidance state machine
      void
      runGuidanceStateMachine(void)
      {
        switch(m_guidance_mode)
        {
          case LONG_STRETCH:
```

```
          guidanceLongStretch();
          break;

        case INCREMENTED_LONG_STRETCH:
          guidanceIncrementedLongStretch();
          break;

        case OPTIMAL_LONG_STRETCH:
          guidanceOptLongStretch();
          break;
        case OWSI:
          guidanceOWSI();
          break;
        default:
          war("Current guidance law not available");
          break;
      }
    }

    //Final mode of state machine
    void
    goToSafeHeight(void)
    {
      if(!m_hasPoint)
      {
        m_hasPoint = true;
        inf("Going to safe height!");
        makePointFromCarp(GOING_TO_SAFE_HEIGHT, true, false, 0,0,100,
m_target.z + m_args.safe_height);

        dispatch_reference();
      }

      if(m_new_FollowRefState)
      {
        m_new_FollowRefState = false;
        if (m_follow_ref.proximity & IMC::FollowRefState::PROX_Z_NEAR)
        {
          m_current_state = IDLE;
          if(!m_args.repeated_testing){
            stopExecution(m_stop_type);
          }
          if(m_args.repeated_testing){
            consume(&m_target);
          }
        }
      }
    }

    //State where the UAV goes to the start point
    void
    goToStartPoint(void)
    {
          double lat, lon, height;
          getCurrentLatLonHeight(&lat, &lon, &height);
          //check speed vector vs carp wind vector and closeness to start point
          if(m_new_FollowRefState)
          {
            m_new_FollowRefState = false;
            if ((m_follow_ref.proximity & IMC::FollowRefState::PROX_XY_NEAR)
                && (m_follow_ref.proximity & IMC::FollowRefState::PROX_Z_NEAR)
&&
                WGS84::distance(m_start_point.lat, m_start_point.lon, 0, lat,
lon, 0) < m_args.accepted_distance_to_start_point)
            {
              if(isParallelish(m_estate, m_carp_ewind, m_args.percent_accurate))
              {
                m_current_state = GOING_TO_DROP_POINT;
                m_follow_ref.proximity = 0;
                m_ref[m_current_state].setTimeStamp();
              }
            }
          }
        }

        //Glide mode for dropping with engine off
        void
        runGlideMode(void)
        {
          if((timer.getMsec() - m_t_start_glide) > 1000 * m_args.glide_time)
          {
            inf("GLIDE MODE finished");
            m_current_state = GOING_TO_SAFE_HEIGHT;
            m_break.op=IMC::Brake::OP_STOP;
            dispatch(m_break);
            m_isGliding = false;
          }
        }

        //Main state machine
        void
        runStateMachine(void)
        {
          if(m_follow_reference_is_running)
          {
            switch(m_current_state)
            {
              case GOING_TO_START_POINT:
                goToStartPoint();
                break;
              case GOING_TO_DROP_POINT:
                runGuidanceStateMachine();
                break;
              case GLIDE_MODE:
                runGlideMode();
                break;
            case GOING_TO_SAFE_HEIGHT:
              goToSafeHeight();
              break;
            default:
              war("Current reference point out of bounds");
              return;

          }
        }
      }

      //Consume target and calculate first CARP and start point
      void
      consume(const IMC::Target* msg)
      {
        if(m_initiated){
          inf("Target received!");

          //Make sure to stop all actions if target is received in mission
          m_break.op=IMC::Brake::OP_STOP;
          dispatch(m_break);
          m_isGliding = false;
          m_smallest_distance = INT_MAX;
          m_previous_distance = INT_MAX;
          m_hasPoint = false;

          //Save wind for non-optimized guidance mode
          m_carp_ewind = m_ewind;
          m_wind_unit_x = m_ewind.x / sqrt(m_ewind.x * m_ewind.x + m_ewind.y *
m_ewind.y);
          m_wind_unit_y = m_ewind.y / sqrt(m_ewind.x * m_ewind.x + m_ewind.y *
m_ewind.y);

          //Save target
          m_target = *msg;
          m_target_received = true;

          //Set target in Beacon class
          m_beacon.set_target(m_target);
          inf("Target set to: %f %f %f", m_beacon.get_target().lat,
m_beacon.get_target().lon, m_beacon.get_target().z);

          //Calculate CARP using wind speed to determine direction (this point
is used to calculate start point for long stretch and used as CARP in non-
optimized guidance mode)
          m_beacon.calculate_CARP_velocity(-m_wind_unit_x * m_args.speed, -
m_wind_unit_y * m_args.speed, 0, m_args.release_height + m_target.z,
m_carp_ewind, m_args.dt, m_args.counter_max);
          inf("CARP has been calculated!");
          inf("CARP: lat: %f, lon: %f, height: %f", m_beacon.get_CARP().lat,
m_beacon.get_CARP().lon, m_beacon.get_CARP().z);

          //Reference for loitering start point
          m_dz.z_units = IMC::Z_HEIGHT;
          makePointFromCarp(GOING_TO_START_POINT, true, false, -m_wind_unit_x,
-m_wind_unit_y, m_args.radius, m_args.release_height + m_target.z);

          //Start point of long stretch
          m_start_point.lat = m_ref[GOING_TO_START_POINT].lat;
          m_start_point.lon = m_ref[GOING_TO_START_POINT].lon;
          m_start_point.z = m_ref[GOING_TO_START_POINT].z.get()->value;

          //Displace the start point one radius to the side
          WGS84::displace(
              m_args.radius * m_wind_unit_y,
              -1 * m_args.radius * m_wind_unit_x ,
              &m_ref[GOING_TO_START_POINT].lat, &m_ref
[GOING_TO_START_POINT].lon);

          //Mission end point
          if(m_guidance_mode == INCREMENTED_LONG_STRETCH){
            m_end_point.z = m_args.release_height;
            m_end_point.lat = m_beacon.get_CARP().lat;
            m_end_point.lon = m_beacon.get_CARP().lon;

            //Displace away from CARP
            WGS84::displace(
                - m_args.end_point_distance_from_carp * m_wind_unit_x,
                - m_args.end_point_distance_from_carp * m_wind_unit_y,
                &m_end_point.lat, &m_end_point.lon);
          }
          if(m_args.direct_to_opt && m_guidance_mode == OPTIMAL_LONG_STRETCH)
            m_current_state = GOING_TO_DROP_POINT;
          else
          {
            m_current_state = GOING_TO_START_POINT;
            inf("Start point has been calculated!");

            inf("Going to start point");
          }
          startExecution();
          dispatch_reference();
        }
        else
        {
          war("New target received and ignored");
        }
      }

      //Check if in air
      void
      consume(const IMC::VehicleMedium* vm)
      {
        m_ground = (vm->medium == IMC::VehicleMedium::VM_GROUND);
      }
      //Check control state
      void
      consume(const IMC::PlanControlState* msg)
      {
        if(msg->state == PlanControlState::PCS_READY && m_isBlocked )
        {
          m_isReady = true;
          m_isBlocked = false;
        }
        else if(msg->state == PlanControlState::PCS_BLOCKED){
          m_isBlocked = true;
        }
      }

      //Get estimated state
      void
      consume(const IMC::EstimatedState* msg)
      {
        m_estate = *msg;
        m_new_EstimatedState = true;

        if(m_wind_received && m_isReady && !m_ground && !m_initiated)
        {
          //m_initiated on first received EstimatedState and Estimated wind
and not blocked and wind received
          m_initiated = true;
          setEntityState(IMC::EntityState::ESTA_NORMAL, Status::CODE_ACTIVE);
          war("Ready for target coordinates");
        }
        runStateMachine();
      }

      //Get usefull entities
      void
      consume(const IMC::EntityList* msg)
      {
        if(m_wind_est_ent == -1)
        {
          TupleList tup(msg->list, "=", ";", true);
          m_wind_est_ent = tup.get("Wind Estimator", -1);
          m_ardupilot_wind_ent = tup.get("Autopilot", -1);
        }
      }

      //Get wind from estimator or ardupilot
      void
      consume(const IMC::EstimatedStreamVelocity* msg)
      {
        if((msg->getSourceEntity() == m_wind_est_ent && m_args.use_wind_est)
|| (!m_args.use_wind_est && msg->getSourceEntity() == m_ardupilot_wind_ent))
        {
          m_ewind = *msg;
          m_wind_received = true;
```

```
  }
}

//Check how the drop mechanism is doing
void
consume(const IMC::PowerChannelState* msg)
{
  if(msg->state == IMC::PowerChannelState::PCS_ON)
  {
    m_pcc.op = 0;
    dispatch(m_pcc);
    uint64_t dt = timer.getMsec() - m_drop_reaction_time;
    err("Delay from drop signal until dropped: %f", dt);
  }
}

//Get follow reference state
void
consume(const IMC::FollowRefState* msg)
{
  m_follow_ref = *msg;
  m_new_FollowRefState = true;
  m_follow_ref.setTimeStamp();

  runStateMachine();

  dispatch_reference();
}

//Check plan control state
void
consume(const IMC::PlanControl* msg)
{
  if (msg->op == IMC::PlanControl::PC_STOP)
  {
    inf("Stop request received");
    m_follow_reference_is_running = false;//Remove from rest of code?
  }
}

//Get battery current
void
consume(const IMC::Current* msg)
{
  m_current = msg->value;
}

void
consume(const IMC::PathControlState* msg){
  m_ct_error = msg->y;
  m_course_error = msg->course_error;
}
```

```
// Start FollowReference maneuver
void
startExecution(void)
{
  if(!m_follow_reference_is_running)
  {
    inf("Starting followReference plan...");
    IMC::PlanControl startPlan;
    startPlan.type = IMC::PlanControl::PC_REQUEST;
    startPlan.op = IMC::PlanControl::PC_START;
    startPlan.plan_id = "drop_plan";
    IMC::FollowReference man;
    man.control_ent = getEntityId();
    man.control_src = getSystemId();
    man.altitude_interval = m_args.altitude_accuracy;
    man.timeout = m_args.connection_timeout;

    IMC::PlanSpecification spec;

    spec.plan_id = "drop_plan";
    spec.start_man_id = "follow_drop";
    IMC::PlanManeuver pm;
    pm.data.set(man);
    pm.maneuver_id = "follow_drop";
    pm.start_actions = maneuverParameters(m_args.bank);
    spec.maneuvers.push_back(pm);
    startPlan.arg.set(spec);
    startPlan.request_id = 0;
    startPlan.flags = 0;

    m_follow_reference_is_running = true;

    dispatch(startPlan);
  }
}

// Stop ongoing FollowReference maneuver
void
stopExecution(uint8_t stop_type)
{
  if(m_follow_reference_is_running)
  {
    inf("Stopping drop plan...");
    IMC::PlanControl stopPlan;
    stopPlan.type = stop_type;
    stopPlan.op = IMC::PlanControl::PC_STOP;
    stopPlan.plan_id = "drop_plan";

    m_follow_reference_is_running = false;
    m_hasPoint = false;
    m_increments = m_args.increments_input;

    dispatch(stopPlan);
    }
  }

  //Make an enum from string
  Modes
  guidance_mode_string_to_enum(char* guidance_mode_input)
  {
    if(strcmp("INCREMENTED_LONG_STRETCH" , guidance_mode_input) == 0)
      return INCREMENTED_LONG_STRETCH;
    else if(strcmp("OPTIMAL_LONG_STRETCH" , guidance_mode_input) == 0)
      return OPTIMAL_LONG_STRETCH;
    else if(strcmp("LONG_STRETCH" , guidance_mode_input) == 0)
      return LONG_STRETCH;
    else if(OWSI)
      return OWSI;
    return  LONG_STRETCH;
  }

  //Check if speed direction is parallell to the wind
  bool
  isParallelish(IMC::EstimatedState estate, IMC::EstimatedStreamVelocity
ewind, fp64_t percentAccurate)
  {
    double stateAngle = atan2(estate.vx, estate.vy);
    double windAngle = atan2(ewind.x, ewind.y);

    double diff = (Angles::normalizeRadian(stateAngle - windAngle));

    //inf("State angle is: %f and wind angle is %f", stateAngle,
windAngle);
    inf("Difference is: %f and has to be bigger than %f", diff, Math::c_pi
* (1 - percentAccurate/100.0));
    if(abs(diff) > Math::c_pi * (1 - percentAccurate/100.0))
      return true;

    return false;
  }

  //Set controller (path or waypoint)
  IMC::MessageList<IMC::Message>
  maneuverParameters(bool bank)
  {
    IMC::MessageList<IMC::Message> setEntityParameters;

    IMC::SetEntityParameters* sep = new IMC::SetEntityParameters();
    sep->name = "Path Control";
    IMC::MessageList<IMC::EntityParameter> entityParameters;
    IMC::EntityParameter* ep = new IMC::EntityParameter();
    ep->name = "Use controller";
    ep->value = bank ? "true" : "false";
    entityParameters.push_back(*ep);
    delete ep;
    sep->params = entityParameters;
  setEntityParameters.push_back(*sep);
  delete sep;

  sep = new IMC::SetEntityParameters();
  sep->name = "Height Control";
  entityParameters.clear();
  ep = new IMC::EntityParameter();
  ep->name = "Active";
  ep->value = bank ? "true" : "false";
  entityParameters.push_back(*ep);
  delete ep;
  sep->params = entityParameters;
  setEntityParameters.push_back(*sep);
  delete sep;

  sep = new IMC::SetEntityParameters();
  sep->name = "Autopilot";
  entityParameters.clear();
  ep = new IMC::EntityParameter();
  ep->name = "Ardupilot Tracker";
  ep->value = bank ? "false" : "true";
  entityParameters.push_back(*ep);
  delete ep;

  ep = new IMC::EntityParameter();
  ep->name = "Formation Flight";
  ep->value = "false";
  entityParameters.push_back(*ep);
  delete ep;
  sep->params = entityParameters;
  setEntityParameters.push_back(*sep);
  delete sep;

  return setEntityParameters;
}

//Drop without checking if motor is off
void
oldDrop(void)
{
  m_pcc.op = 1;
  dispatch(m_pcc);
  //Wait before sending 0? TEST IT!
  m_pcc.op = 0;
  dispatch(m_pcc);
}

//Drop beacon
void
drop(double distance_to_point)
{
  if(m_current < m_args.max_current and m_isGliding)
  {
          inf("Current is OK!");
          m_pcc.name = "drop";
          m_pcc.op = 1;

          m_drop_reaction_time = timer.getMsec();
          dispatch(m_pcc);

          inf("Dropped %f meters away from optimum!", distance_to_point);
          inf("With a speed vector in [N: %f], [E: %f], [D: %f] away from
optimum", m_estate.vx - m_beacon.get_CARP().vx, m_estate.vy -
m_beacon.get_CARP().vy, m_estate.vz - m_beacon.get_CARP().vz);
        }
        else{
          err("Current is NOT OK!");
        }
      }

      //Make a point depending on CARP
      void
      makePointFromCarp(int state, bool directFlag, bool startFlag, double
x_unit_speed, double y_unit_speed, double radius, double height){
        m_ref[state].radius = radius; //Small radius to make UAV go straight:
DANGER! DANGER! //20 is minimum
        m_ref[state].lat = m_beacon.get_CARP().lat;
        m_ref[state].lon = m_beacon.get_CARP().lon;

        m_dz.value = height;
        m_ref[state].speed.set(m_dspeed);
        m_ref[state].z.set(m_dz);

        if(directFlag){
          m_ref[state].flags = Reference::FLAG_LOCATION |
Reference::FLAG_RADIUS | Reference::FLAG_Z  | Reference::FLAG_DIRECT;

          WGS84::displace(
              - m_args.end_point_distance_from_carp * x_unit_speed,
              - m_args.end_point_distance_from_carp * y_unit_speed,
              &m_ref[state].lat, &m_ref[state].lon);
        }
        else if(startFlag){
          m_ref[state].flags = Reference::FLAG_LOCATION |
Reference::FLAG_RADIUS | Reference::FLAG_Z  | Reference::FLAG_START_POINT;

          WGS84::displace(
              m_args.start_point_distance_from_carp * x_unit_speed,
              m_args.start_point_distance_from_carp * y_unit_speed,
              &m_ref[state].lat, &m_ref[state].lon);
        }
        else{
          m_ref[state].flags = Reference::FLAG_LOCATION |
Reference::FLAG_RADIUS | Reference::FLAG_Z;

          WGS84::displace(
            - m_args.end_point_distance_from_carp * x_unit_speed,
            - m_args.end_point_distance_from_carp * y_unit_speed,
            &m_ref[state].lat, &m_ref[state].lon);
      }
    }

    //Get current lat lon and height
    void
    getCurrentLatLonHeight(fp64_t *lat, fp64_t *lon, fp64_t *height){
      *lat = m_estate.lat, *lon = m_estate.lon, *height = m_estate.height;
      WGS84::displace(m_estate.x, m_estate.y, m_estate.z, lat, lon, height);
    }

    //Check how far you are from the CARP in a certain number of seconds
    double
    distanceByTime(double lat, double lon, double height, double time){
      WGS84::displace( //Displacing forward (drop time) times speed because
of delay
          m_estate.vx * (time),
          m_estate.vy * (time),
          m_estate.vz * (time),    //m_estate can be replaced with
m_beacon.get_CARP.vx etc.
          &lat, &lon, &height);

      return WGS84::distance(m_beacon.get_CARP().lat, m_beacon.get_CARP
().lon, 0, lat,  lon, 0);
    }

    //Make two points over the CARP together making a line
    void
    makeStraightLineOverCarp(double vx_unit, double vy_unit){

      //For testing
      //vx_unit = 0;
      //vy_unit = 0;

      //Make and dispatch point A in a line for LOS to follow
      makePointFromCarp(GOING_TO_DROP_POINT, false, true, -vx_unit, -
vy_unit, 0, m_target.z + m_args.release_height);
      dispatch_reference();

      //Make and dispatch point B in a line for LOS to follow
      makePointFromCarp(GOING_TO_DROP_POINT, false, false, -vx_unit, -
vy_unit, 0, m_target.z + m_args.release_height);
      dispatch_reference();
    }

    //Check if you are close to the CARP
    bool
    isCloseToPoint(double distance_to_point){
      return (((distance_to_point < m_args.drop_error) && (distance_to_point
> m_previous_distance)) || distance_to_point < m_args.speed/(10));
    }
```

```
      //Check if the motor has been off for a certain time
      bool
      motorShutDownTimeHasRun(){
        return (m_t_start_glide - timer.getMsec()) > m_args.glide_time * 1000;
      }

      //Check if you have passed the CARP
      void
      checkForFailure(){
        if(m_new_FollowRefState && (m_follow_ref.proximity &
IMC::FollowRefState::PROX_XY_NEAR)
            && (m_follow_ref.proximity & IMC::FollowRefState::PROX_Z_NEAR)
            && (m_ref[m_current_state].getTimeStamp() + 10000) <
m_follow_ref.getTimeStamp())
        {
          m_new_FollowRefState = false;
          war("FAILURE: CARP missed! Was as close as: %f",
m_smallest_distance);

          m_break.op=IMC::Brake::OP_STOP;
          dispatch(m_break);
          m_isGliding = false;
          m_smallest_distance = INT_MAX;
          m_previous_distance = INT_MAX;
          m_t_start_glide = timer.getMsec(); //=0?

          //m_stop_type = IMC::PlanControl::PC_FAILURE;
          m_current_state = GOING_TO_SAFE_HEIGHT;
        }
        else
        {
          m_new_FollowRefState = false;
        }
      }

      //Dispatch current reference
      void
      dispatch_reference(void)
      {
        if(m_current_state > IDLE and m_current_state < LAST_STATE)
        {
          dispatch(m_ref[m_current_state]);
        }
      }

      //! Main loop.
      void
      onMain(void)
      {
        while (!stopping())
        {
         /* uint64_t start = timer.getMsec();
```

```
          m_beacon.optimal_CARP(100, m_ewind, m_estate, m_args.dt,
                                m_args.counter_max, m_args.opt_circle,
                                m_args.opt_points, m_W_vel, m_args.w_pos,
m_args.glide_time);
          uint64_t dt = timer.getMsec() - start;
          inf("TIME FOR OPT: %u msec", dt);*/
          waitForMessages(1.0);
        }
      }
    };
  }
}

DUNE_TASK
```

H.3

Task.cpp

```
// Copyright 2007-2014 Universidade do Porto - Faculdade de Engenharia      *

// DUNE headers.
#include <DUNE/DUNE.hpp>
//#include <fstream>
//#include <iostream>

using namespace std;

namespace Actuators
{
  namespace LogicOutput
  {
    namespace BBB
    {
      using DUNE_NAMESPACES;

      struct Arguments
      {
        //! Pin to control
        int pin;
        //! Initial value
        int init;
        //! Name.
        std::string name;
      };

      struct Task: public DUNE::Tasks::Task
      {
        IMC::PowerChannelState m_status;
        GPIO* m_out;
        //! Constructor.
        //! @param[in] name task name.
        //! @param[in] ctx context.
        Arguments m_args;
        Task(const std::string& name, Tasks::Context& ctx):
          DUNE::Tasks::Task(name, ctx),
          m_out(NULL)
        {
          param("Pin", m_args.pin)
          .defaultValue("2")
          .description("Pin to control.");

          param("Initial Output", m_args.init)
          .defaultValue("0")
          .description("Initial output on pin.");

          param("Name", m_args.name)
          .defaultValue("drop")
          .description("Name of PowerChannel.");

          bind<PowerChannelControl>(this);
        }
```

```
//! Update internal state with new parameter values.
void
onUpdateParameters(void)
{
  war("started act");
  m_status.name = m_args.name;
}

//! Reserve entity identifiers.
void
onEntityReservation(void)
{
}

//! Resolve entity names.
void
onEntityResolution(void)
{
}

//! Acquire resources.
void
onResourceAcquisition(void)
{
  m_out = new GPIO(m_args.pin);
  m_out->setDirection(GPIO::GPIO_DIR_OUTPUT);
  m_out->setValue(m_args.init);
  inf("Drop mechanism ready!");
  setEntityState(IMC::EntityState::ESTA_NORMAL, Status::CODE_ACTIVE);
}

//! Initialize resources.
void
onResourceInitialization(void)
{
  //m_out->setValue(true);
}

//! Release resources.
void
onResourceRelease(void)
{
}

void
consume(const IMC::PowerChannelControl* msg)
{
  if(!msg->name.compare(m_args.name)){
    if(msg->op == 0 || msg->op == 1)
    {
      //Turn off/on the drop mechanism
      m_out->setValue(msg->op);
```

```
            if (msg->op == 0){
              m_status.state = IMC::PowerChannelState::PCS_OFF;
              dispatch(m_status);
            }
            if (msg->op == 1){
              m_status.state = IMC::PowerChannelState::PCS_ON;
              dispatch(m_status);
            }
          }
          else
          {
            war("Received a non drop op");
          }
        }
      }

      //! Main loop.
      void
      onMain(void)
      {
        while (!stopping())
        {
          waitForMessages(10.0);
        }
      }
    };
  }
}
}

DUNE_TASK
```

H.4

```
// Copyright 2007-2014 Universidade do Porto - Faculdade de Engenharia      *

// DUNE headers.
#include <DUNE/DUNE.hpp>
#include <mavlink/ardupilotmega/mavlink.h>
#include <time.h>

using namespace std;

namespace Navigation
{
  namespace WindEstimator
  {
    using DUNE_NAMESPACES;

    struct Arguments
    {
      string auto_pilot_type;
      int sample_window_size;
      double trustedlim;
      double q_multi;
    };

    struct Task: public DUNE::Tasks::Task
    {

      //! Type definition for Arduino packet handler.
      typedef void (Task::* PktHandler)(const mavlink_message_t* msg);
      typedef std::map<int, PktHandler> PktHandlerMap;
      PktHandlerMap m_mlh;

      Math::Matrix m_C, m_P, m_Q, m_R, m_w, m_G, m_R_bn, m_vn, m_vb, m_vr,
m_d, m_S, m_K;
      double m_e, m_alpha, m_beta,  m_air_speed, m_dt, timer;
      bool hasEstate, measurementsIstrusted;
      int m_n_samples;

      IMC::EstimatedState m_estate;
      IMC::EstimatedStreamVelocity m_wind_estimated;
      IMC::EstimatedStreamVelocity m_wind_at_the_moment;

      //! Constructor.
      //! @param[in] name task name.
      //! @param[in] ctx context.
      Arguments m_args;
      Task(const std::string& name, Tasks::Context& ctx):
        DUNE::Tasks::Task(name, ctx)
      {
        param("Auto Pilot", m_args.auto_pilot_type)
        .defaultValue("ardupilot")
        .description("Tuning parameters are set depending on the autopilot");

        param("Sample Window Size", m_args.sample_window_size)
  .defaultValue("1000")
  .description("Sample window size");

  param("Trusted Limit", m_args.trustedlim)
  .defaultValue("0")
  .description("Trusted observability gramian limit");

  param("Q Size", m_args.q_multi)
  .defaultValue("1")
  .description("Q size multiplier");

  bind<EstimatedState>(this);
  bind<IndicatedSpeed>(this);
}


//! Update internal state with new parameter values.
void
onUpdateParameters(void)
{
  hasEstate = false;
  measurementsIstrusted = false;

  double adjp = 1, adjq = m_args.q_multi, adjr = 1;

  double p[] = {0.01*adjp, 0.01*adjp, 0.001*adjp, 1e-4*adjp};
  m_P = Matrix(p,4);
  double r[] = {1*adjr};
  m_R = Matrix(r, 1, 1);
  double q[] = {1e-3*adjq, 1e-3*adjq,  1e-4*adjq,  1e-8*adjq};
  m_Q = Matrix(q,4);
  double w[] = {0,0,0,1};
  m_w = Matrix(w,4,1);
  double g[] = {0,0,0,0};
  m_G = Matrix(g, 4);
  double d[] = {1, 0, 0};
  m_d = Matrix(d,1,3);

  m_e = 0;
  m_alpha = 0;
  m_beta = 0;
  m_dt = 0;
  m_air_speed = 0;

  timer = Clock().get();

  setEntityState(IMC::EntityState::ESTA_NORMAL, Status::CODE_ACTIVE);
}

void
handleIas(void)
{
  m_dt = Clock().get() - timer; //First might be large
```

```
        timer = Clock().get();

        if(m_n_samples<m_args.sample_window_size){
          m_n_samples++;
        }


        if(hasEstate){
          double r_bn[] = {-m_estate.phi, -m_estate.theta, -m_estate.psi};
          double vb[] = {m_estate.u, m_estate.v, m_estate.w};
          double vn[] ={m_estate.vx, m_estate.vy, m_estate.vz};
          m_R_bn = Matrix(r_bn, 3, 1).toQuaternion().toDCM();
          m_vb = Matrix(vb, 3, 1);
          m_vn = Matrix(vn,3,1);

          double tas[] = {m_air_speed};
//          m_C = horseCat(m_d * transpose(m_R_bn), Matrix(tas,1,1));
          m_C = m_d * transpose(m_R_bn);
          m_C = m_C.horzCat(Matrix(tas,1,1));

          m_P = m_P + m_Q * pow(m_dt,2);
          m_S = m_C * m_P * transpose(m_C) + m_R;
          m_K = m_P * transpose(m_C) * inverse(m_S);
          m_P = (Matrix(4) - (m_K * m_C)) * m_P * transpose(Matrix(4) - m_K *
m_C) + m_K * m_R * transpose(m_K);
          m_P = (m_P * transpose(m_P)) / 2;

          m_e = (m_d * m_vb - (m_C * m_w)).element(0,0);

          m_w = m_w + (m_K * m_e);

          m_vr = m_vb - transpose(m_R_bn) * m_w.get(0, 2, 0, 0);

          //Find normalized observability gramian
          m_G = m_G + m_dt * transpose(m_C) * m_C / m_n_samples;

          double temp[] = {m_air_speed,0,0};

          m_wind_at_the_moment.x = (m_vn - (transpose(m_R_bn) * Matrix
(temp,3,1))).element(0,0);
          m_wind_at_the_moment.y = (m_vn - (transpose(m_R_bn) * Matrix
(temp,3,1))).element(1,0);
          m_wind_at_the_moment.z = (m_vn - (transpose(m_R_bn) * Matrix
(temp,3,1))).element(2,0);

          m_wind_estimated.x = +m_w.element(0,0);
          m_wind_estimated.y = +m_w.element(1,0);
          m_wind_estimated.z = +m_w.element(2,0);

          dispatch(m_wind_at_the_moment);
          dispatch(m_wind_estimated);

          if((m_G.detr() > m_args.trustedlim) and !measurementsIstrusted){
```

```
          measurementsIstrusted = true;
          war("Measurement trusted");
        }
        else if(!(m_G.detr() > m_args.trustedlim) and measurementsIstrusted)
{
          measurementsIstrusted = false;
          war("Measurement not trusted");
        }

        //Find AoA and SSA
        m_alpha = atan(m_vr.element(2,0) / m_vr.element(0,0)) * 180 /
Math::c_pi;
        m_beta = asin(m_vr.element(1,0) / m_vr.norm_2()) * 180 / Math::c_pi;
      }
    }

//      Matrix
//      horseCat(Matrix a, Matrix b)
//      {
//        if(a.rows() != b.rows()){
//          throw Exception("invalid index");
//        }
//        const int length =(a.rows()+b.rows())*a.columns();
//        double ret[length];
//        int counter = 0;
//        for(int i = 0; i<a.rows(); i++){
//          for(int j= 0; j<a.columns(); j++){
//            ret[counter] = a.element(i,j);
//            counter++;
//          }
//          for(int j= 0; j<b.columns(); j++){
//            ret[counter] = b.element(i,j);
//            counter++;
//          }
//        }
//        return Matrix(ret,a.rows(),a.columns()+b.columns());
//      }

    // Use Matrix operator<< for printing.

//      void
//      printMatrix(Matrix m)
//      {
//        printf("[TEST PRINTOUT]\n");
//        for(int i = 0; i<m.rows(); i++ ){
//          for(int j = 0; j<m.columns();j++){
//            printf("%f ", m.element(i,j));
//          }
//          printf("\n");
//        }
//      }
//
//      void
//      printMatrix(Matrix m, string name)
//      {
//        printf("[");
//        printf(name.c_str());
//        printf("]\n");
//        for(int i = 0; i<m.rows(); i++ ){
//          for(int j = 0; j<m.columns();j++){
//            printf("%f ", m.element(i,j));
//          }
//          printf("\n");
//        }
//      }

      //! Reserve entity identifiers.
      void
      onEntityReservation(void)
      {
        m_wind_at_the_moment.setSourceEntity(reserveEntity("Current wind"));
      }

      //! Resolve entity names.
      void
      onEntityResolution(void)
      {
      }

      void
      consume(const IMC::EstimatedState* msg)
      {
        m_estate = *msg;
        hasEstate = true;
      }

      void
      consume(const IMC::IndicatedSpeed* msg)
      {
        m_air_speed = msg->value;
        handleIas();
      }

      //! Main loop.
      void
      onMain(void)
      {
        while (!stopping())
        {
          waitForMessages(1.0);
        }
      }
    };
  }
}
```

```
DUNE_TASK
```

H.5

```
// Copyright 2007-2015 Universidade do Porto - Faculdade de Engenharia      *

// ISO C++ 98 headers.
#include <cmath>

// DUNE headers.
#include <DUNE/DUNE.hpp>

namespace Control
{
  namespace Path
  {
    namespace SMC
    {
      using DUNE_NAMESPACES;

      struct Arguments
      {
        //! Maximum bank angle - Defined by aircaft structural, navigation
        //! or control constraints
        double max_bank;
        double lookahead;
        double err_gain;
        double err_dot_gain;
        double err_int_gain;
        double roll_tc;
        //!
        bool use_controller;
      };

      struct Task: public DUNE::Control::PathController
      {
        Arguments m_args;
        IMC::DesiredRoll m_bank;
        double m_airspeed;
        double m_bank_lim;
        double m_lookahead, m_lookahead_sq;
        double m_y_int;

        Task(const std::string& name, Tasks::Context& ctx):
          DUNE::Control::PathController(name, ctx),
          m_airspeed(0.0),
          m_bank_lim(0.0),
          m_lookahead(0.0),
          m_lookahead_sq(0.0),
          m_y_int(0.0)
        {
          param("Lookahead", m_args.lookahead)
          .defaultValue("100.0")
          .description("Lookahead distance");

          param("Error P Gain", m_args.err_gain)
          .defaultValue("1.0")
          .description("Turn rate gain for control");

          param("Error D Gain", m_args.err_dot_gain)
          .defaultValue("1.0")
          .description("Turn rate gain for control");

          param("Error I Gain", m_args.err_int_gain)
          .defaultValue("1.0")
          .description("Turn rate gain for control");

          param("Roll Time Const", m_args.roll_tc)
          .defaultValue("1.0")
          .description("Turn rate gain for control");

          param("Maximum Bank", m_args.max_bank)
          .units(Units::Degree)
          .minimumValue("5")
          .maximumValue("45")
          .defaultValue("30")
          .description("Limit for absolute value of output bank angle
reference");

          param("Use controller", m_args.use_controller)
          .visibility(Tasks::Parameter::VISIBILITY_USER)
          .scope(Tasks::Parameter::SCOPE_MANEUVER)
          .defaultValue("false")
          .description("Use this controller for maneuver");

          bind<IMC::IndicatedSpeed>(this);
       }

        void
        onUpdateParameters(void)
        {
          PathController::onUpdateParameters();

          m_bank_lim = Angles::radians(m_args.max_bank);
          m_lookahead = m_args.lookahead;
          m_lookahead_sq = m_lookahead * m_lookahead;
        }

        void
        onPathActivation(void)
        {
          if (!m_args.use_controller)
            return;
          // Activate bank (roll) controller.
          enableControlLoops(IMC::CL_ROLL);
        }

        void
        onPathStartup(const IMC::EstimatedState& state, const TrackingState&
ts)
```

```
        {
          (void) state;
          (void) ts;
          m_y_int = 0;
        }

        void
        consume(const IMC::IndicatedSpeed* airspeed)
        {
          m_airspeed = airspeed->value;
        }

        void
        step(const IMC::EstimatedState& state, const TrackingState& ts)
        {
          spew("Step start");

          if (!m_args.use_controller)
            return;

          //! Check if airspeed is larger than zero
          if (m_airspeed <= 0)
          {
            war("No waypoint tracking control update: Airspeed <= 0!");
            return;
          }

          double speed_g = ts.speed; //std::sqrt(ts.track_vel.y *
ts.track_vel.y
                                    //+ ts.track_vel.x * ts.track_vel.x);

          double chi = std::atan2(ts.track_vel.y, ts.track_vel.x);
          double chi_dot = Math::c_gravity * std::tan(state.phi) / speed_g;

          double y = ts.track_pos.y;
          double y_sq = y * y;
//          double y_dot = - speed_g * y / std::sqrt(m_lookahead_sq + y_sq);
          double y_dot = - speed_g * std::sin(chi);

          double gamma_dot = 0;
          if (ts.loitering)
          {
            gamma_dot = (ts.loiter.clockwise ? 1 : -1) * ts.track_vel.x /
ts.loiter.radius;
          }

          //! LOS
          double chi_d = -std::atan(y/m_lookahead);
          double chi_d_dot = gamma_dot -(m_lookahead/(m_lookahead_sq + y_sq))
* y_dot;

          //! ILOS
//          double y_y_int_sq = (y + m_args.err_int_gain * m_y_int) * (y +
m_args.err_int_gain * m_y_int);
//          double y_int_dot = m_lookahead * y / (y_y_int_sq +
m_lookahead_sq);
//          m_y_int += ts.delta * y_int_dot;
//          double chi_d = -std::atan((y + m_args.err_int_gain * m_y_int)/
m_lookahead);
//          double chi_d_dot = -(y_dot + m_args.err_int_gain * y_int_dot) /
(m_lookahead * (y_y_int_sq / m_lookahead_sq + 1));

          double chi_err = chi - chi_d;
          double chi_err_dot = chi_dot - chi_d_dot;

          double u = -m_args.err_dot_gain * chi_err_dot;
          u -= m_args.err_gain * std::tanh(chi_err);

          double bank_d = u * speed_g * std::cos(state.phi) * m_args.roll_tc /
Math::c_gravity;

          //! Output - Bank angle command, constrained
          m_bank.value = trimValue(bank_d, -m_bank_lim, m_bank_lim);

          // Send to bus
          dispatch(m_bank);
          spew("DesiredBank = %3.2f deg", Math::Angles::degrees
(m_bank.value));
        }
      };
    }
  }
}

DUNE_TASK
```

H.6

```
 * Beacon.hpp

#ifndef BEACON_HPP_
#define BEACON_HPP_

#include <DUNE/DUNE.hpp>
#include "Point.hpp"
#include <limits.h>

class Beacon
{

private:
  DUNE::IMC::Target target;
  Point CARP;
  fp32_t mass;
  fp32_t b;
  fp32_t g;
  double t;

public:
  Beacon()
{
    fp32_t rho = 1.341; // air density at -10 degrees celsius, according to
Simen Fuglaas
    fp32_t C_D = 0.5;   // drag coefficient of smooth sphere
    fp32_t r = 0.05;    // 0.05 meters in radius. Assume the beacon is a
sphere
    fp32_t A = M_PI*r*r;
    b = 0.5*C_D*rho*A;  //damping constant
    mass = 0.2;         // mass
    g = 9.81;           //gravity constant
    CARP = Point();
    t = 0;
};

  void calculate_CARP_speed(double speed, double release_height,
DUNE::IMC::EstimatedStreamVelocity wind, fp32_t dt, int counter_max)
  {
    fp64_t x = 0, y = 0, z = 0, d = 0, vx = 0, vy = 0, vz = 0, x_copy = 0,
y_copy = 0;
    fp32_t wind_speed = sqrt(wind.x * wind.x + wind.y * wind.y);
    vx = -wind.x / wind_speed * speed;
    vy = -wind.y / wind_speed * speed;
    vz = 0;
    x_copy = x;
    y_copy = y;
    z = release_height;
    int counter = 0;
    while(z > target.z and counter < counter_max)
    {
      counter ++;
      fp32_t v_rel_abs = sqrt(((vx - wind.x) * (vx - wind.x)) + ((vy - wind.y)
* (vy - wind.y)) + ((vz - wind.z) * (vz-wind.z)));
      x += vx*dt;
      y += vy*dt;
      z -= vz*dt;
      vx += -b / mass * (vx - wind.x) * v_rel_abs * dt;
      vy += -b / mass * (vy - wind.y) * v_rel_abs * dt;
      vz += (-b / mass * (vz - wind.z) * v_rel_abs + g) * dt;
    }
    //time to reach ground
    t = counter * dt;
    // Calculate position of release in (x,y)-plane
    double dx = x - x_copy;
    double dy = y - y_copy;
    CARP.speed = speed;
    CARP.lat = target.lat;
    CARP.lon = target.lon;

    WGS84::displace(-dx,-dy, &CARP.lat, &CARP.lon);

    CARP.z = target.z + release_height;
    CARP.vx = vx / wind_speed * speed;
    CARP.vy = vy / wind_speed * speed;
    CARP.vz = 0;
  };

  void calculate_CARP_velocity(fp64_t vx, fp64_t vy, fp64_t vz, double
release_height, DUNE::IMC::EstimatedStreamVelocity wind, fp32_t dt, int
counter_max)
  {
    CARP.speed = sqrt(vx * vx + vy * vy + vz * vz);
    CARP.vx = vx;
    CARP.vy = vy;
    CARP.vz = 0;

    fp64_t x = 0, y = 0, z = 0, d = 0, x_copy = 0, y_copy = 0;
    z = release_height;

    int counter = 0;
    while(z > target.z and counter < counter_max)
    {
      counter ++;
      fp32_t v_rel_abs = sqrt(((vx - wind.x) * (vx - wind.x)) + ((vy - wind.y)
* (vy - wind.y)) + ((vz - wind.z) * (vz - wind.z)));
      x += vx * dt;
      y += vy * dt;
      z -= vz * dt;
      vx += -b / mass * (vx - wind.x) * v_rel_abs* dt;
      vy += -b / mass * (vy - wind.y) * v_rel_abs * dt;
      vz += (-b / mass * (vz - wind.z) * v_rel_abs + g) * dt;
    }

    //time to reach ground
    t = counter * dt;
    // Calculate position of release in (x,y)-plane
    double dx = x - x_copy;
    double dy = y - y_copy;
    CARP.lat = target.lat;
    CARP.lon = target.lon;

    WGS84::displace(-dx, -dy, &CARP.lat, &CARP.lon);

    CARP.z = release_height;
  };

  void optimal_CARP(double release_height, DUNE::IMC::EstimatedStreamVelocity
wind, DUNE::IMC::EstimatedState estate, fp32_t dt,
      int counter_max, double rad_of_circle, int num_of_points, Matrix
weight_velocity, double weight_position, double glide_time)
  {
    double best_sum = 10000;
    double test_sum = 0;
    Point optimal_carp;
    Matrix V_new;
    double v_now[] = {estate.vx, estate.vy, 0};
    Matrix V_now = Matrix(v_now, 3, 1);
    double speed = sqrt(estate.vx * estate.vx + estate.vy * estate.vy);
    //Speed minus speed lost while gliding (t*C_d*A*rho)
    double v_tot[] = {speed - glide_time * .1 * (2 * .10 * 3.14 + .80 * 2 * 2)
* 1.225 , 0, 0};
    Matrix V_tot = Matrix(v_tot, 3, 1);

    double angle;
    double lat = estate.lat, lon = estate.lon, height = estate.height;
    WGS84::displace(estate.x, estate.y, estate.z, &lat, &lon, &height);

    double angle_to_target = atan2(target.lon - lon, target.lat - lat); //lat
lon switch?
    double body_speed_angle = atan2(estate.vy, estate.vx);

    //Turn v_body about z axis for accurate weighing (mirror about angle to
target)
    double a = 2*(angle_to_target - body_speed_angle);
    double rot_v_about_z[] = {cos(a), -sin(a), 0, sin(a),
        cos(2*(angle_to_target + body_speed_angle)), 0, 0, 0, 1};
    V_now = Matrix(rot_v_about_z,3,3) * V_now ;

    for(int i = -1 * floor(num_of_points / 2.0); i < ceil(num_of_points /
2.0); i++){
      //Rotate into currently tested CARP velocity
      angle = (angle_to_target) + i * rad_of_circle / num_of_points;
      double rotz[] = {cos(angle), -sin(angle), 0, sin(angle), cos(angle), 0,
0, 0, 1};
      V_new = Matrix(rotz, 3, 3) * V_tot;
      calculate_CARP_velocity(V_new.element(0,0), V_new.element(1,0),
V_new.element(2,0), release_height, wind, dt, counter_max);
```

```
      //Test if currently best
      test_sum =   (weight_velocity * abs(V_new - V_now)).element(0, 0) +
weight_position *  WGS84::distance(CARP.lat,CARP.lon  , release_height,
lat,lon, release_height);

      if(test_sum < best_sum){
        best_sum = test_sum;
        optimal_carp = CARP;
      }
    }
    CARP = optimal_carp;
  };

  Point get_CARP()
  {
    return CARP;
  };
  double get_t(){
    return t;
  };
  DUNE::IMC::Target get_target()
  {
    return target;
  };
  void set_CARP(Point CARP_)
  {
    this->CARP = CARP_;
  };
  void set_target(DUNE::IMC::Target target_)
  {
    this->target = target_;
  };
};



#endif /* BEACON_HPP_ */
```

H.7

Enums.hpp

```
 * DropOnTargetEnums.hpp

#ifndef DROPONTARGETENUMS_HPP_
#define DROPONTARGETENUMS_HPP_

using DUNE_NAMESPACES;

enum Modes {LONG_STRETCH = 0, INCREMENTED_LONG_STRETCH, OPTIMAL_LONG_STRETCH,
OWSI};

enum States {IDLE=-1, GOING_TO_START_POINT, GOING_TO_DROP_POINT,
GOING_TO_SAFE_HEIGHT, GLIDE_MODE, LAST_STATE};

#endif /* DROPONTARGETENUMS_HPP_ */
```

H.8

```
/*
 * PWM_generator.c
 *
 * Created: 30.09.2014 10:31:11
 *  Author: Vegard
 */

#define PWMPIN PB0
#define INPUTPIN PB1
#define F_CPU 8000000

#include <avr/io.h>
#include <util/delay.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>
#include <stdlib.h>


// Routines to set and clear bits (used in the sleep code)
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

int makeSignal = 0;

ISR(PCINT0_vect)             // interrupt service routine
{
        //Change interupt flag
        makeSignal = 1;
        return;
}

void setup_watchdog(int ii) {

        int bb;
        //int ww;
        if (ii > 9 ) ii=9;
        bb=ii & 7;
        if (ii > 7) bb|= (1<<5);
        bb|= (1<<WDCE);
        //ww=bb;

        MCUSR &= ~(1<<WDRF);
        // start timed sequence
        WDTCR |= (1<<WDCE) | (1<<WDE);
        // set new watchdog timeout value
        WDTCR = bb;
        WDTCR |= _BV(WDIE);
}
```

```
void sleep(){
        // sleep mode is set here
        set_sleep_mode(SLEEP_MODE_PWR_DOWN);
        sleep_enable();

        // System actually sleeps here
        sleep_mode();

        // System continues execution here when watchdog timed out
        sleep_disable();
}


void PWM_init(){
        //Enable input on the inputpin
        DDRB = 0x00;

        //Set the inputpin low
        PORTB |= (0 << INPUTPIN);

        // pin change mask: listen to portb bit 2
        PCMSK |= (1<<PCINT1);

        // enable PCINT interrupt
        GIMSK |= (1<<PCIE);

        // enable all interrupts
        sei();

        //Enable output on the outputpin
        DDRB |= (1 << PWMPIN);

        //Clear OC0A on Compare Match , using fast PWM mode
        TCCR0A |= (1 << COM0A1) | (1 << WGM01) | (1 << WGM00);

        //Fast PWM mode with 0xFF = TOP, Clock select 64 prescaler
        TCCR0B |= (0 << WGM02) | (1 << CS01) | (1 << CS00);

        //Init the PWM signal to be compared with the counter value
        OCR0A = 0;

        // approximately xxx seconds sleep
        setup_watchdog(2);
}

void PWM_run(){
        uint64_t value;
        int mini=0;
        for(value = mini; value < 50; value++){
                OCR0A = value;
                _delay_ms(2);
        }

        for(value = 50; value > mini; value --){
                OCR0A = value;
                _delay_ms(2);
        }
}

int main(void)
{
        PWM_init();

    while(1)
    {
         if(makeSignal == 1){
                        PWM_run();
                        makeSignal = 0;
                }
                else{
                        OCR0A = 0;
                        DDRB |= (0 << PWMPIN);
                        sleep();
                        DDRB |= (1 << PWMPIN);
                }
    }
}


//SIRIS:
/*
#include <avr/io.h>
#define PWMPIN PB0
#define INPUTPIN PB1
void PWM_init(){
        //Enable input on the inputpin
        DDRB = 0x00;
        //Set the inputpin low
        PORTB |= (0 << INPUTPIN);
        //Enable output on the outputpin
        DDRB |= (1 << PWMPIN);
        //Clear OC0A on Compare Match , using fast PWM mode
        TCCR0A |= (1 << COM0A1) | (1 << WGM01) | (1 << WGM00);
        //Fast PWM mode with 0xFF = TOP, Clock selecy 64 prescaler
        TCCR0B |= (0 << WGM02) | (1 << CS01) | (1 << CS00);
        //Init the PWM signal to be compared with the counter value
        OCR0A = 0;
}
void PWM_run(){
        uint64_t value;
        uint64_t sleeper;
        for(value = 0; value < 50; value++){
                OCR0A = value;
                sleeper = 0;
                while(sleeper < 300){
                        sleeper++;
                }
        }
        for(value = 50; value > 0; value --){
                OCR0A = value;
                sleeper = 0;
                while(sleeper < 300){
                        sleeper++;
                }
        }
}
int main(void)
{
        PWM_init();
        while(1)
        {
                if(PINB & (1 << INPUTPIN)){
                        PWM_run();
                }else{
                        OCR0A = 0;
                }
        }
}*/
```

Appendix I

MATLAB code

I.1 Free Fall Initiation Script

```
 clear all;
 close all;
 clc;
%%
% Define constants
iter = 10;
% Physical properties of object
A = pi*0.05^2; % Cross-sectional area of a sphere
C_D = 0.47; % Drag constant for a sphere
m = 0.2; % Mass of the object
rho = 1.225; % Air density at +15 degrees Celsius
b = 0.5*A*C_D*rho;
g = 9.81;
% Wind velocity
w = 5;
w_angle = 180;
v_w = [w*cos(w_angle);w*sin(w_angle);0];
w_uncertainty = 0.5;
```

```matlab
w_angle_uncert = 10;
w_magn_range = ...
    (w*(1-w_uncertainty)):w*w_uncertainty/iter:(w*(1+w_uncertainty));
w_angle_range = ...
    (w_angle-w_angle_uncert):(w_angle_uncert/iter):(w_angle+w_angle_uncert);
% Initial velocity
v_init = 28;
% Initial position
x_0_bx = 0;
x_0_by = 0;
x_0_alt = 50;
mu_pos = [pi x_0_by];
bx_uncert = 5;
by_uncert = 5;
bx_range = 0:bx_uncert/iter/2:bx_uncert;
by_range = -by_uncert:by_uncert/iter:by_uncert;
[bX1,bX2] = meshgrid(bx_range, by_range);
sigma_pos = [1 0; 0 1];
norm_pos = mvnpdf([bX1(:) bX2(:)], mu_pos, sigma_pos);
norm_pos = reshape(norm_pos,length(bx_range), length(by_range));
figure
surf(bx_range, by_range, norm_pos);
% Simulation time
dt = 0.01;
tspan = 0:dt:6;
% Initialize ground level matrix 1000x1000 m
Impact = zeros(1000,1000);
% Iterations
tot_iter = length(w_magn_range)*length(w_angle_range);
i_ = 1;
% Create waitbar to show progress
h = waitbar(0,'wait...');
% Simulate position given varying wind
for i=1:(length(w_magn_range))
    for j=1:(length(w_angle_range))

```

```
        w = w_magn_range(i);
        w_angle = w_angle_range(j);
        w_angle = w_angle/180*pi;
        v_w = [w*cos(w_angle);w*sin(w_angle);0];

        v_0 = [v_init+w*cos(w_angle);v_init+w*sin(w_angle);0];
        x_0 = [v_0;x_0_bx;x_0_by;x_0_alt];

        % Solve differential system
        [t,v] = ode45(@(t,v) freeFallFunc2(t,v,v_w,b,m,g), tspan, x_0);
        figure(98)
        plot3(v(:,4), v(:,5),v(:,6))
        hold on;grid on;xlabel('x');ylabel('y');zlabel('z');
        % Position
        x_incr_reso = v(end,4:6)*1;
        x_rounded = round(x_incr_reso(:));
        Impact(x_rounded(1),x_rounded(2)) =+ norm_pos(i,j);
        % Update waitbar
        i_ = i_+1;
        waitbar(i_/tot_iter,h);
    end
end
close(h);
x_start = 35;
y_start = x_start;
x_end = 55;
y_end = x_end;
points = 20;
figure
surf(linspace(x_start,x_end,points),linspace(y_start,...
    y_end,points),Impact(x_start:x_end-1,y_start:y_end-1))
```

I.2 Free Fall Function

```
function dv = freeFallFunc2(t,v,v_w,b,m,g)
% Define the function used to model the free fall of the object
    if v(6) ≤ 0
        dv = [-(b/m)*(v(1)-v_w(1))*norm(v(1:3)-v_w);
        -(b/m)*(v(2)-v_w(2))*norm(v(1:3)-v_w);
        -(b/m)*(v(3)-v_w(3))*norm(v(1:3)-v_w)-g;
         0;
        0;
        0];
    else
          dv = [-(b/m)*(v(1)-v_w(1))*norm(v(1:3)-v_w);
        -(b/m)*(v(2)-v_w(2))*norm(v(1:3)-v_w);
        -(b/m)*(v(3)-v_w(3))*norm(v(1:3)-v_w)-g;
        v(1);
        v(2);
        v(3)];
    end
end
```

I.3 Plot RTK and Pixhawk Position Script

```
%%INIT
clear;
load('Data rtk piksi');
%%START
figure(1);
clf;
hold on;
title('3D Position for RTK and Pixhawk');
itend = [length(EstimatedState.x),length(RtkFix.n)];
diff = [EstimatedState.x(itend(1))-RtkFix.n(itend(2)), ...
    EstimatedState.y(itend(1))-RtkFix.e(itend(2)), ...
    EstimatedState.z(itend(1))-RtkFix.d(itend(2))];
plot3(EstimatedState.x - diff(1), EstimatedState.y - diff(2), ...
    EstimatedState.z - diff(3));
plot3(RtkFix.n, RtkFix.e, RtkFix.d);
xlabel('N');
ylabel('E');
zlabel('D');


%%START
figure(2);
clf;
hold on;
title('NE Position for RTK and Pixhawk');
plot(EstimatedState.x - diff(1), EstimatedState.y - diff(2));
plot(RtkFix.n, RtkFix.e);
xlabel('N');
ylabel('E');

%%START
figure(3);
clf;
hold on;
title('ND Position for RTK and Pixhawk');
plot(EstimatedState.x - diff(1), EstimatedState.z - diff(3));
plot(RtkFix.n, RtkFix.d);
xlabel('N');
ylabel('D');

measurediff=zeros(3,length(RtkFix.n));
measureerr=zeros(3,length(RtkFix.n));

for i = 1:length(RtkFix.n)
    prevtdiff=10000;
    leastdiff=10000;
    for j = 1:length(EstimatedState.x)
```

```
        tdiff = abs(RtkFix.timestamp(i)-EstimatedState.timestamp(j));
        if tdiff<leastdiff
            leastdiff=tdiff;
        elseif tdiff > prevtdiff
            measurediff(1,i) = EstimatedState.x(j) - diff(1)-RtkFix.n(i);
            measureerr(1,i) = abs(EstimatedState.x(j) - ...
                diff(1)-RtkFix.n(i));
            measurediff(2,i) = EstimatedState.y(j) - diff(2)-RtkFix.e(i);
            measureerr(2,i) = abs(EstimatedState.y(j) - ...
                diff(2)-RtkFix.e(i));
            measurediff(3,i) = EstimatedState.z(j) - diff(3)-RtkFix.d(i);
            measureerr(3,i) = abs(EstimatedState.z(j) - ...
                diff(3)-RtkFix.d(i));
            break;
        end
        prevtdiff = tdiff;
    end
end

Mn= mean(measurediff(1,:));
Me= mean(measurediff(2,:));
Md= mean(measurediff(3,:));
Vn= var(measurediff(1,:));
Ve= var(measurediff(2,:));
Vd= var(measurediff(3,:));

Man= mean(measureerr(1,:));
Mae= mean(measureerr(2,:));
Mad= mean(measureerr(3,:));
Van= var(measureerr(1,:));
Vae= var(measureerr(2,:));
Vad= var(measureerr(3,:));

%%SAVE
%saveas(figure(2), 'My Master/Report - Master/figures/rtkpixpos_ne', ...
    'png');
```

```
%saveas(figure(3), 'My Master/Report - Master/figures/rtkpixpos_nd', ...
    'png');

%%
```

I.4 Plot Wind Estimator 2 m/s Results Script

```
clear;
load('Data 2.mat');
ent = zeros(1,25);
entcount = 1;

for i = 1:length(EstimatedStreamVelocity.x)
    newent = true;
    for j = 1:length(ent)
        if(EstimatedStreamVelocity.src_ent(i) == ent(j))
            newent = false;
        end
    end
    if(newent)
        ent(entcount) = EstimatedStreamVelocity.src_ent(i);
        entcount = entcount +1;
    end
end

x = zeros(entcount,length(EstimatedStreamVelocity.x));
y = x;
z = x;
count = ones(1,entcount);
for i = 1:length(EstimatedStreamVelocity.x)
    for j = 1:entcount
        if EstimatedStreamVelocity.src_ent(i) == ent(j)
            x(j,count(j)) = EstimatedStreamVelocity.x(i);
            y(j,count(j))= EstimatedStreamVelocity.y(i);
            z(j,count(j))= EstimatedStreamVelocity.z(i);
            count(j) = count(j) +1;
            continue;
        end
    end
end

hold on;
%%
num = 0;
for j = length(EstimatedStreamVelocity.x):-1:1
    if x(1,j) ≠ 0
        num = j;
        break;
    end
end
figure(1);
clf;
hold on;
title('Pixhawk Wind Estimation, 2 m/s Simulated')
xlabel('Timestamp');
ylabel('m/s');
plot(EstimatedStreamVelocity.timestamp(1:num),  x(1,1:num), '--');
plot(EstimatedStreamVelocity.timestamp(1:num),  y(1,1:num),':');
plot(EstimatedStreamVelocity.timestamp(1:num), z(1,1:num));
legend('n','e','d')
num = 0;


for j = length(EstimatedStreamVelocity.x):-1:1
    if x(2,j) ≠ 0
        num = j;
        break;
    end
end
```

```
figure(2);
clf;
hold on;
title('Wind Measurment, 2 m/s Simulated')
xlabel('Timestamp');
ylabel('m/s');
plot(EstimatedStreamVelocity.timestamp(1:num),  x(2,1:num), '--');
plot(EstimatedStreamVelocity.timestamp(1:num),  y(2,1:num),':');
plot(EstimatedStreamVelocity.timestamp(1:num), z(2,1:num));
legend('n','e','d')
num = 0;


for j = length(EstimatedStreamVelocity.x):-1:1
    if x(3,j) ≠ 0
        num = j;
        break;
    end
end
figure(3);
clf;
hold on;
title('Wind Estimate, 2 m/s Simulated')
xlabel('Timestamp');
ylabel('m/s');
plot(EstimatedStreamVelocity.timestamp(1:num),  x(3,1:num), '--');
plot(EstimatedStreamVelocity.timestamp(1:num),  y(3,1:num),':');
plot(EstimatedStreamVelocity.timestamp(1:num), z(3,1:num));
legend('n','e','d')
saveas(figure(3), 'My Master/Report - Master/figures/2ms_est','png')
saveas(figure(1), 'My Master/Report - Master/figures/2ms_pix','png')
saveas(figure(2), 'My Master/Report - Master/figures/2ms_measure','png')

num=120;%rest is zero
actual_wind(1:num) = 2;
pmn=mean(actual_wind-x(1,1:num));
```

```
pvn=var(actual_wind-x(1,1:num));
actual_wind(1:num) = 0;
pme=mean(actual_wind-y(1,1:num));
pve=var(actual_wind-y(1,1:num));
pmd=mean(actual_wind-z(1,1:num));
pvd=var(actual_wind-z(1,1:num));

actual_wind(1:num) = 2;
pman= mean(abs(actual_wind-x(1,1:num)));
pvan= var(abs(actual_wind-x(1,1:num)));
actual_wind(1:num) = 0;
pmae=mean(abs(actual_wind-y(1,1:num)));
pvae=var(abs(actual_wind-y(1,1:num)));
pmad=mean(abs(actual_wind-z(1,1:num)));
pvad=var(abs(actual_wind-z(1,1:num)));

num=990;%rest is zero
actual_wind(1:num) = 2;
mn=mean(actual_wind-x(3,1:num));
vn=var(actual_wind-x(3,1:num));
actual_wind(1:num) = 0;
me=mean(actual_wind-y(3,1:num));
ve=var(actual_wind-y(3,1:num));
md=mean(actual_wind-z(3,1:num));
vd=var(actual_wind-z(3,1:num));

actual_wind(1:num) = 2;
man= mean(abs(actual_wind-x(3,1:num)));
van= var(abs(actual_wind-x(3,1:num)));
actual_wind(1:num) = 0;
mae=mean(abs(actual_wind-y(3,1:num)));
vae=var(abs(actual_wind-y(3,1:num)));
mad=mean(abs(actual_wind-z(3,1:num)));
vad=var(abs(actual_wind-z(3,1:num)));
```

I.5 Plot Wind Estimator 5 m/s Results Script

```
clear;
load('Data 5.mat');
ent = zeros(1,25);
entcount = 1;

for i = 1:length(EstimatedStreamVelocity.x)
    newent = true;
    for j = 1:length(ent)
        if(EstimatedStreamVelocity.src_ent(i) == ent(j))
            newent = false;
        end
    end
    if(newent)
        ent(entcount) = EstimatedStreamVelocity.src_ent(i);
        entcount = entcount +1;
    end
end

x = zeros(entcount,length(EstimatedStreamVelocity.x));
y = x;
z = x;
count = ones(1,entcount);
for i = 1:length(EstimatedStreamVelocity.x)
    for j = 1:entcount
        if EstimatedStreamVelocity.src_ent(i) == ent(j)
            x(j,count(j)) = EstimatedStreamVelocity.x(i);
            y(j,count(j))= EstimatedStreamVelocity.y(i);
            z(j,count(j))= EstimatedStreamVelocity.z(i);
            count(j) = count(j) +1;
            continue;
        end
    end
end
```

```

hold on;
%%
num = 0;
for j = length(EstimatedStreamVelocity.x):-1:1
    if x(1,j) ≠ 0
        num = j;
        break;
    end
end
figure(1);
clf;
hold on;
title('Wind Measurment, 5 m/s Simulated')
xlabel('Timestamp');
ylabel('m/s');
plot(EstimatedStreamVelocity.timestamp(1:num),  x(1,1:num), '--');
plot(EstimatedStreamVelocity.timestamp(1:num),  y(1,1:num),':');
plot(EstimatedStreamVelocity.timestamp(1:num), z(1,1:num));
legend('n','e','d')
num = 0;


for j = length(EstimatedStreamVelocity.x):-1:1
    if x(2,j) ≠ 0
        num = j;
        break;
    end
end
figure(2);
clf;
hold on;
title('Wind Estimate, 5 m/s Simulated')
xlabel('Timestamp');
ylabel('m/s');
plot(EstimatedStreamVelocity.timestamp(1:num),  x(2,1:num), '--');
plot(EstimatedStreamVelocity.timestamp(1:num),  y(2,1:num),':');
plot(EstimatedStreamVelocity.timestamp(1:num), z(2,1:num));
legend('n','e','d')
num = 0;




for j = length(EstimatedStreamVelocity.x):-1:1
    if x(3,j) ≠ 0
        num = j;
        break;
    end
end
figure(3);
clf;
hold on;
title('Pixhawk Wind Estimation, 5 m/s Simulated')
xlabel('Timestamp');
ylabel('m/s');
plot(EstimatedStreamVelocity.timestamp(1:num),  x(3,1:num), '--');
plot(EstimatedStreamVelocity.timestamp(1:num),  y(3,1:num),':');
plot(EstimatedStreamVelocity.timestamp(1:num), z(3,1:num));
legend('n','e','d')
saveas(figure(1), 'My Master/Report - Master/figures/5ms_measure','png')
saveas(figure(2), 'My Master/Report - Master/figures/5ms_est','png')
saveas(figure(3), 'My Master/Report - Master/figures/5ms_pix','png')

num=140;%rest is zero
actual_wind(1:num) = 5;
pmn=mean(actual_wind-x(3,1:num));
pvn=var(actual_wind-x(3,1:num));
actual_wind(1:num) = 0;
pme=mean(actual_wind-y(3,1:num));
pve=var(actual_wind-y(3,1:num));
pmd=mean(actual_wind-z(3,1:num));
```

```
pvd=var(actual_wind-z(3,1:num));

actual_wind(1:num) = 5;
pman= mean(abs(actual_wind-x(3,1:num)));
pvan= var(abs(actual_wind-x(3,1:num)));
actual_wind(1:num) = 0;
pmae=mean(abs(actual_wind-y(3,1:num)));
pvae=var(abs(actual_wind-y(3,1:num)));
pmad=mean(abs(actual_wind-z(3,1:num)));
pvad=var(abs(actual_wind-z(3,1:num)));


num=1180;%rest is zero
actual_wind(1:num) = 5;
mn=mean(actual_wind-x(2,1:num));
vn=var(actual_wind-x(2,1:num));
actual_wind(1:num) = 0;
me=mean(actual_wind-y(2,1:num));
ve=var(actual_wind-y(2,1:num));
md=mean(actual_wind-z(2,1:num));
vd=var(actual_wind-z(2,1:num));

actual_wind(1:num) = 5;
man= mean(abs(actual_wind-x(2,1:num)));
van= var(abs(actual_wind-x(2,1:num)));
actual_wind(1:num) = 0;
mae=mean(abs(actual_wind-y(2,1:num)));
vae=var(abs(actual_wind-y(2,1:num)));
mad=mean(abs(actual_wind-z(2,1:num)));
vad=var(abs(actual_wind-z(2,1:num)));
```

I.6 Process LS Path SIL Results Script

```
clear;
load('Data lots of ls.mat');
ent = zeros(1,25);
entcount = 1;

for i = 1:length(Distance.value)
    newent = true;
    for j = 1:length(ent)
        if(Distance.src_ent(i) == ent(j))
            newent = false;
        end
    end
    if(newent)
        ent(entcount) = Distance.src_ent(i);
        entcount = entcount +1;
    end
    if i==500
        break;
    end
end
if i==500
    x = zeros(entcount,500);
else
    x = zeros(entcount,round(length(Distance.value)/entcount));
end

count = ones(1,entcount);
for i = 1:(entcount*length(x(1,:)))
    for j = 1:entcount
        if Distance.src_ent(i) == ent(j)
            x(j,count(j)) = Distance.value(i);
            count(j) = count(j) +1;
            continue;
        end
    end
```

```
end

average=[mean(x(1,:)),mean(x(2,:)),mean(x(3,:))];
[xe, ye, ze] = ellipsoid(average(1),average(2),average(3), ...
    sqrt(var(x(1,:))+ var(x(2,:))+ var(x(3,:))),sqrt(var(x(1,:))+ ...
    var(x(2,:))+ var(x(3,:))),sqrt(var(x(1,:))+ var(x(2,:))+ ...
    var(x(3,:))),30);
[xr, yr, zr] = ellipsoid(0,0,0, .85*2, .85*2, .85*2 ,30);

figure(1);
clf;
hold on;
title('LS release accuracy plot in body frame')
xlabel('x');
ylabel('y');
zlabel('z');
scatter3(x(1,:),x(2,:),x(3,:), 'Fill');
scatter3(average(1),average(2),average(3), 'Fill')
scatter3(0,0,0, 'Fill')
surf(xe,ye,ze, 'FaceAlpha', 0.2)
surf(xr,yr,zr, 'FaceAlpha', 0.2)
legend('Release point','Average','CARP', 'MRSE')

figure(2);
clf;
hold on;
title('LS path release accuracy XY-plot in body frame')
xlabel('x');
ylabel('y');
scatter(x(1,:),x(2,:), 20, 'Fill');
scatter(average(1),average(2), 'Fill')
scatter(0,0, 'Fill')
[X,Y]=ell(average(1),average(2),sqrt(var(x(1,:))+ var(x(2,:))), ...
    sqrt(var(x(1,:))+ var(x(2,:))),0,100);
plot(X,Y);
alpha(0)
```

```
legend('Release point','Average','CARP', 'DRMS')

figure(3);
clf;
hold on;
title('LS path release accuracy XZ-plot in body frame')
xlabel('x');
ylabel('z');
scatter(x(1,:),x(3,:), 20,'Fill');
scatter(average(1),average(3), 'Fill')
scatter(0,0, 'Fill')
[X,Y]=ell(average(1),average(3),sqrt(var(x(1,:))+ ...
    var(x(3,:))),sqrt(var(x(1,:))+ var(x(3,:))),0,100);
plot(X,Y);
alpha(0)
legend('Release point','Average','CARP', 'DRMS')


mx= average(1);
my= average(2);
mz= average(3);
vx= var(x(1,:));
vy=var(x(2,:));
vz=var(x(3,:));

max= mean(abs(x(1,:)));
may= mean(abs(x(2,:)));
maz= mean(abs(x(3,:)));
vax= var(abs(x(1,:)));
vay=var(abs(x(2,:)));
vaz=var(abs(x(3,:)));

mvx= mean(x(4,:));
mvy= mean(x(5,:));
mvz= mean(x(6,:));
vvx= var(x(4,:));
```

```
vvy=var(x(5,:));
vvz=var(x(6,:));

mavx= mean(abs(x(4,:)));
mavy= mean(abs(x(5,:)));
mavz= mean(abs(x(6,:)));
vavx= var(abs(x(4,:)));
vavy=var(abs(x(5,:)));
vavz=var(abs(x(6,:)));

saveas(figure(2), 'Report - Master/figures/ls_xy','png')
saveas(figure(3), 'Report - Master/figures/ls_xz','png')
```

I.7 Process Optimal Path SIL Results Script

```
clear;
load('Data lots of optimal.mat');
ent = zeros(1,25);
entcount = 1;

for i = 1:length(Distance.value)
    newent = true;
    for j = 1:length(ent)
        if(Distance.src_ent(i) == ent(j))
            newent = false;
        end
    end
    if(newent)
        ent(entcount) = Distance.src_ent(i);
        entcount = entcount +1;
    end
end

```

```
if i==500
    x = zeros(entcount,500);
else
    x = zeros(entcount,round(length(Distance.value)/entcount));
end

count = ones(1,entcount);
for i = 1:(entcount*length(x(1,:)))
    for j = 1:entcount
        if Distance.src_ent(i) == ent(j)
            x(j,count(j)) = Distance.value(i);
            count(j) = count(j) +1;
            continue;
        end
    end
end

average=[mean(x(1,:)),mean(x(2,:)),mean(x(3,:))];
[xe, ye, ze] = ellipsoid(average(1),average(2),average(3), ...
    sqrt(var(x(1,:))+ var(x(2,:))+ var(x(3,:))),sqrt(var(x(1,:))+ ...
    var(x(2,:))+ var(x(3,:))),sqrt(var(x(1,:))+ var(x(2,:))+ ...
    var(x(3,:))),30);
[xr, yr, zr] = ellipsoid(0,0,0, .85*2, .85*2, .85*2 ,30);

figure(1);
clf;
hold on;
title('Optimal path release accuracy plot in body frame')
xlabel('x');
ylabel('y');
zlabel('z');
scatter3(x(1,:),x(2,:),x(3,:), 'Fill');
scatter3(average(1),average(2),average(3), 'Fill')
scatter3(0,0,0, 'Fill')
surf(xe,ye,ze, 'FaceAlpha', 0.2)
surf(xr,yr,zr, 'FaceAlpha', 0.2)
```

```
legend('Release point','Average','CARP', 'MRSE', 'CARP release distance')

figure(2);
clf;
hold on;
title('Optimal path release accuracy XY-plot in body frame')
xlabel('x');
ylabel('y');
scatter(x(1,:),x(2,:), 20, 'Fill');
scatter(average(1),average(2), 'Fill')
scatter(0,0, 'Fill')
[X,Y]=ell(average(1),average(2),sqrt(var(x(1,:))+ var(x(2,:))), ...
    sqrt(var(x(1,:))+ var(x(2,:))),0,100);
plot(X,Y);
alpha(0)
legend('Release point','Average','CARP', 'DRMS')

figure(3);
clf;
hold on;
title('Optimal path release accuracy XZ-plot in body frame')
xlabel('x');
ylabel('z');
scatter(x(1,:),x(3,:),20, 'Fill');
scatter(average(1),average(3), 'Fill')
scatter(0,0, 'Fill')
[X,Y]=ell(average(1),average(3),sqrt(var(x(1,:))+ ...
    var(x(3,:))),sqrt(var(x(1,:))+ var(x(3,:))),0,100);
plot(X,Y);
alpha(0)
legend('Release point','Average','CARP', 'DRMS')


mx= average(1);
my= average(2);
mz= average(3);
```

```
vx= var(x(1,:));
vy=var(x(2,:));
vz=var(x(3,:));

max= mean(abs(x(1,:)));
may= mean(abs(x(2,:)));
maz= mean(abs(x(3,:)));
vax= var(abs(x(1,:)));
vay=var(abs(x(2,:)));
vaz=var(abs(x(3,:)));

mvx= mean(x(4,:));
mvy= mean(x(5,:));
mvz= mean(x(6,:));
vvx= var(x(4,:));
vvy=var(x(5,:));
vvz=var(x(6,:));

mavx= mean(abs(x(4,:)));
mavy= mean(abs(x(5,:)));
mavz= mean(abs(x(6,:)));
vavx= var(abs(x(4,:)));
vavy=var(abs(x(5,:)));
vavz=var(abs(x(6,:)));

saveas(figure(2), 'Report - Master/figures/optimal_xy','png')
saveas(figure(3), 'Report - Master/figures/optimal_xz','png')
```

I.8 Process OWSI Path SIL Results Script

```
clear;
load('Data 500 owsi.mat');
ent = zeros(1,25);
```

```
entcount = 1;

for i = 1:length(Distance.value)
    newent = true;
    for j = 1:length(ent)
        if(Distance.src_ent(i) == ent(j))
            newent = false;
        end
    end
    if(newent)
        ent(entcount) = Distance.src_ent(i);
        entcount = entcount +1;
    end
end

x = zeros(entcount,round(length(Distance.value)/entcount));

count = ones(1,entcount);
for i = 1:length(Distance.value)
    for j = 1:entcount
        if Distance.src_ent(i) == ent(j)
            x(j,count(j)) = Distance.value(i);
            count(j) = count(j) +1;
            continue;
        end
    end
end

average=[mean(x(1,:)),mean(x(2,:)),mean(x(3,:))];
[xe, ye, ze] = ellipsoid(average(1),average(2),average(3), ...
    sqrt(var(x(1,:))+ var(x(2,:))+ var(x(3,:))),sqrt(var(x(1,:))+ ...
    var(x(2,:))+ var(x(3,:))),sqrt(var(x(1,:))+ var(x(2,:))+ ...
    var(x(3,:))),30);
[xr, yr, zr] = ellipsoid(0,0,0, .85*2, .85*2, .85*2 ,30);

figure(1);
```

```
clf;
hold on;
title('OWSI release accuracy plot in body frame')
xlabel('x');
ylabel('y');
zlabel('z');
scatter3(x(1,:),x(2,:),x(3,:), 'Fill');
scatter3(average(1),average(2),average(3), 'Fill')
scatter3(0,0,0, 'Fill')
surf(xe,ye,ze, 'FaceAlpha', 0.2)
surf(xr,yr,zr, 'FaceAlpha', 0.2)
legend('Release point','Average','CARP', 'MRSE')

figure(2);
clf;
hold on;
title('OWSI path release accuracy XY-plot in body frame')
xlabel('x');
ylabel('y');
scatter(x(1,:),x(2,:),20, 'Fill');
scatter(average(1),average(2), 'Fill')
scatter(0,0, 'Fill')
[X,Y]=ell(average(1),average(2),sqrt(var(x(1,:))+ var(x(2,:))), ...
    sqrt(var(x(1,:))+ var(x(2,:))),0,100);
plot(X,Y);
alpha(0)
legend('Release point','Average','CARP', 'DRMS')

figure(3);
clf;
hold on;
title('OWSI path release accuracy XZ-plot in body frame')
xlabel('x');
ylabel('z');
scatter(x(1,:),x(3,:),20, 'Fill');
scatter(average(1),average(3), 'Fill')
scatter(0,0, 'Fill')
[X,Y]=ell(average(1),average(3),sqrt(var(x(1,:))+ ...
    var(x(3,:))),sqrt(var(x(1,:))+ var(x(3,:))),0,100);
plot(X,Y);
alpha(0)
legend('Release point','Average','CARP', 'DRMS')


mx= average(1);
my= average(2);
mz= average(3);
vx= var(x(1,:));
vy=var(x(2,:));
vz=var(x(3,:));

max= mean(abs(x(1,:)));
may= mean(abs(x(2,:)));
maz= mean(abs(x(3,:)));
vax= var(abs(x(1,:)));
vay=var(abs(x(2,:)));
vaz=var(abs(x(3,:)));

mvx= mean(x(4,:));
mvy= mean(x(5,:));
mvz= mean(x(6,:));
vvx= var(x(4,:));
vvy=var(x(5,:));
vvz=var(x(6,:));

mavx= mean(abs(x(4,:)));
mavy= mean(abs(x(5,:)));
mavz= mean(abs(x(6,:)));
vavx= var(abs(x(4,:)));
vavy=var(abs(x(5,:)));
vavz=var(abs(x(6,:)));

```

```
saveas(figure(2), 'Report - Master/figures/owsi_xy','png')
saveas(figure(3), 'Report - Master/figures/owsi_xz','png')
```

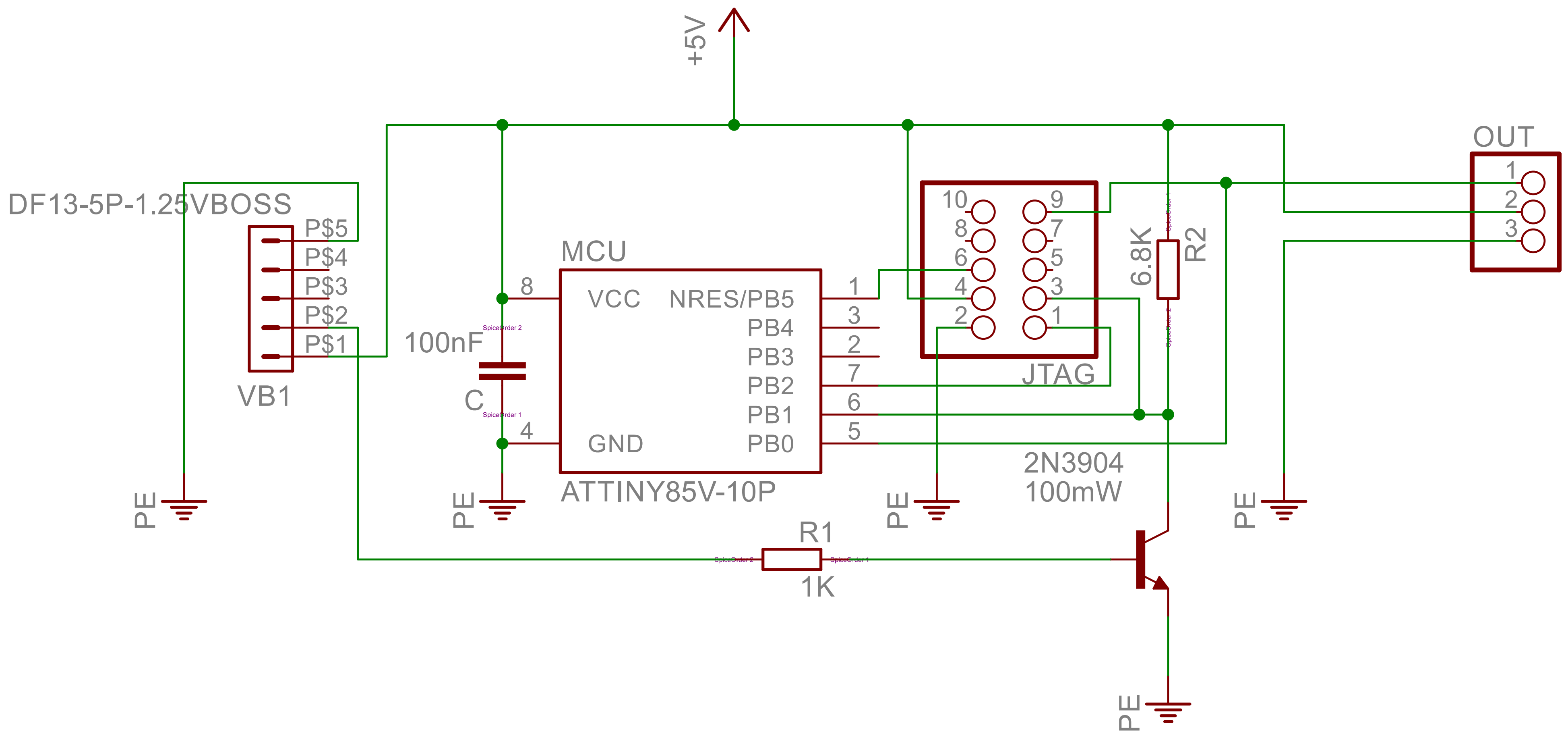
I.9 Draw Ellipse Function

```
function [X,Y] = ell(x, y, a, b, angle, s)
    %# This functions returns points to draw an ellipse
    %#
    %#  @param x     X coordinate
    %#  @param y     Y coordinate
    %#  @param a     Semimajor axis
    %#  @param b     Semiminor axis
    %#  @param angle Angle of the ellipse (in degrees)
    %#

    narginchk(5, 6);
    if nargin<6, s = 36; end

    beta = -angle * (pi / 180);
    sinbeta = sin(beta);
    cosbeta = cos(beta);

    alpha = linspace(0, 360, s)' .* (pi / 180);
    sa = sin(alpha);
    ca = cos(alpha);

    X = x + (a * ca * cosbeta - b * sa * sinbeta);
    Y = y + (a * ca * sinbeta + b * sa * cosbeta);

    if nargout==1, X = [X Y]; end
end
```

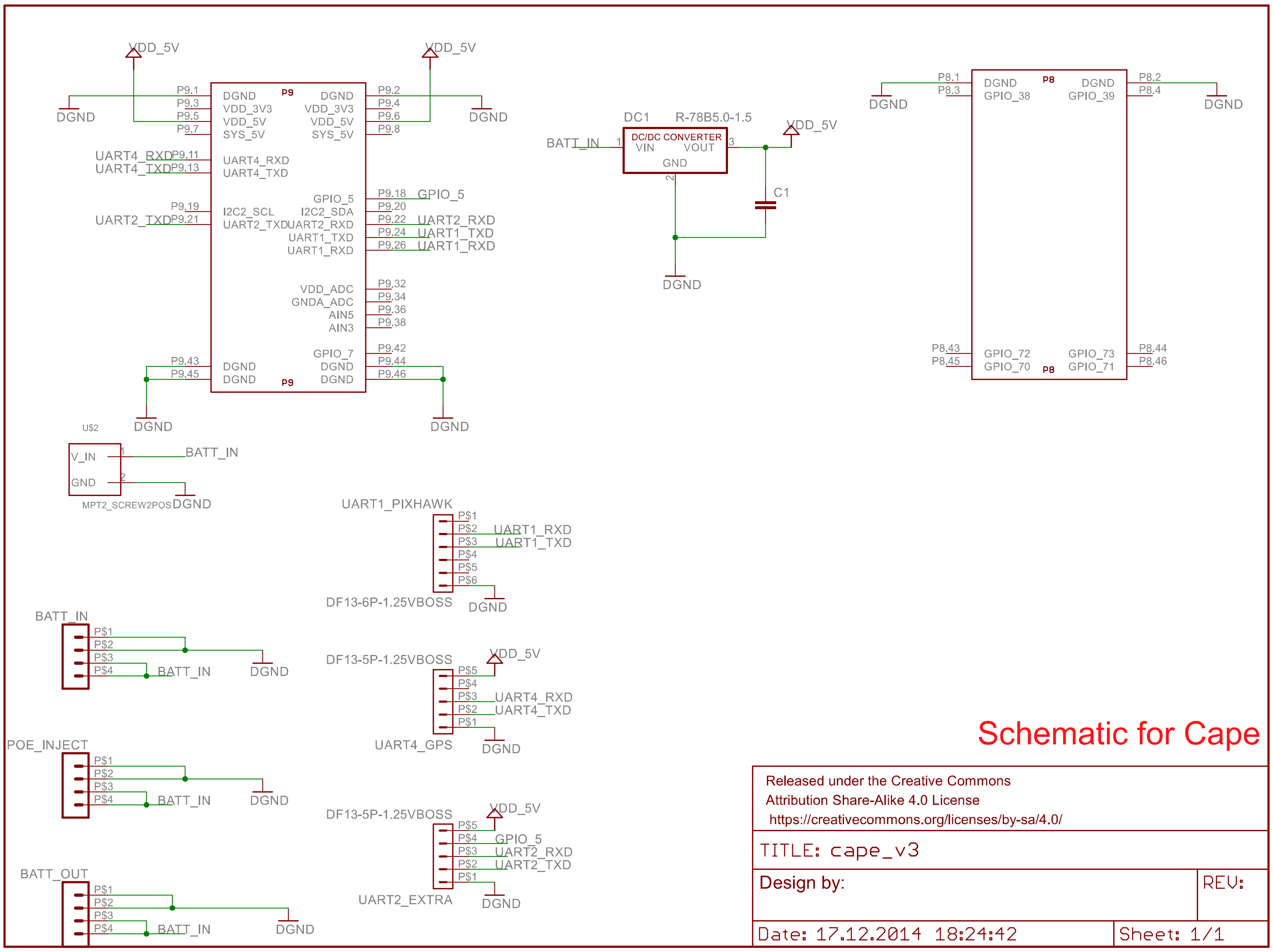
Appendix J

Circuit Schematics

PWM generator schematic



17.12.2014 18:06:15 f=2.00 F:\Google Drive\Skole\Master\Accurate drop of payload using the X8 fixed-wing UAV\PWM\PWMgen Eagle\PWMgen.sch (Sheet: 1/1)



Schematic for Cape

TITLE: cape_v3

Design by:

REV:

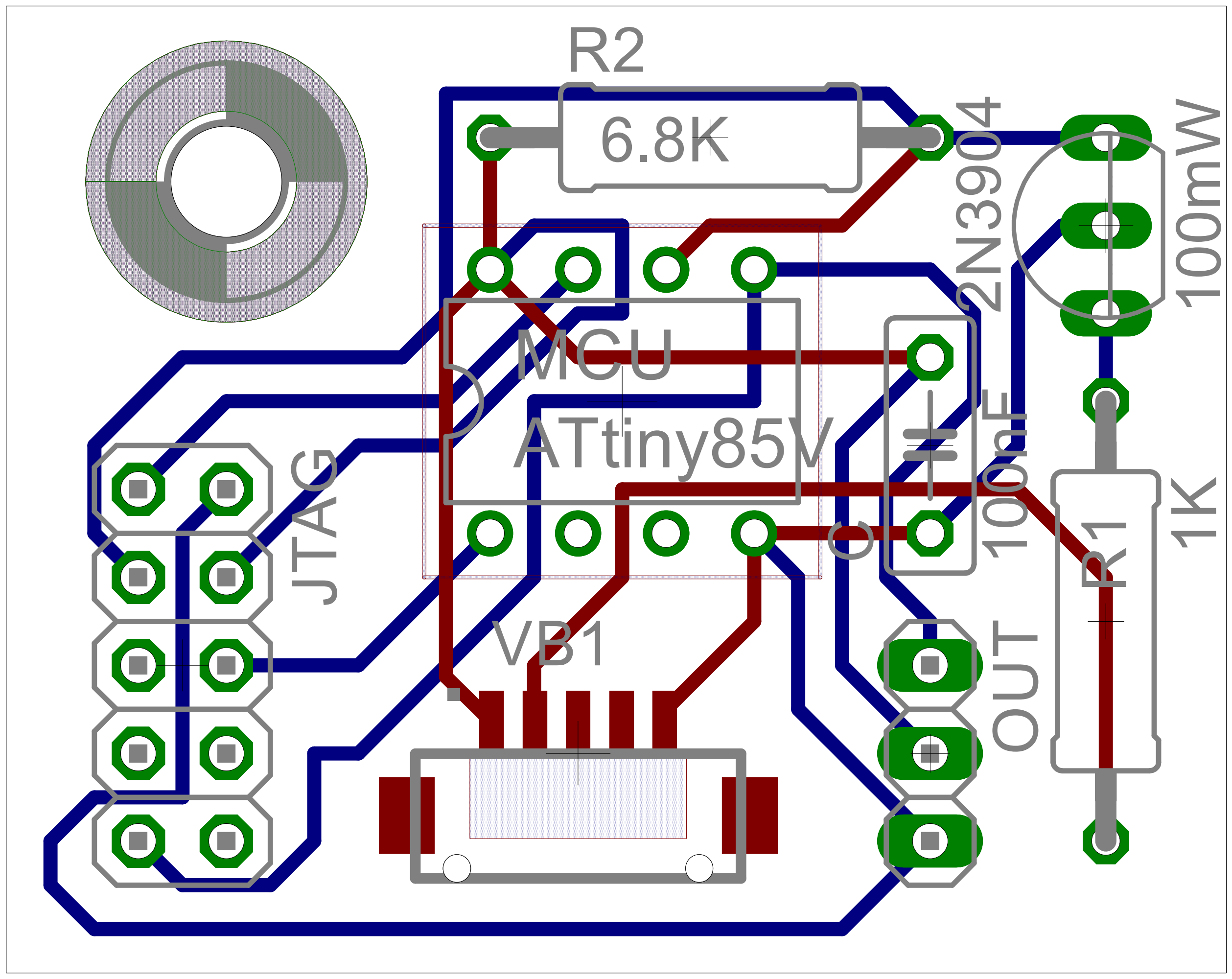
Date: 17.12.2014 18:24:42

Sheet: 1/1

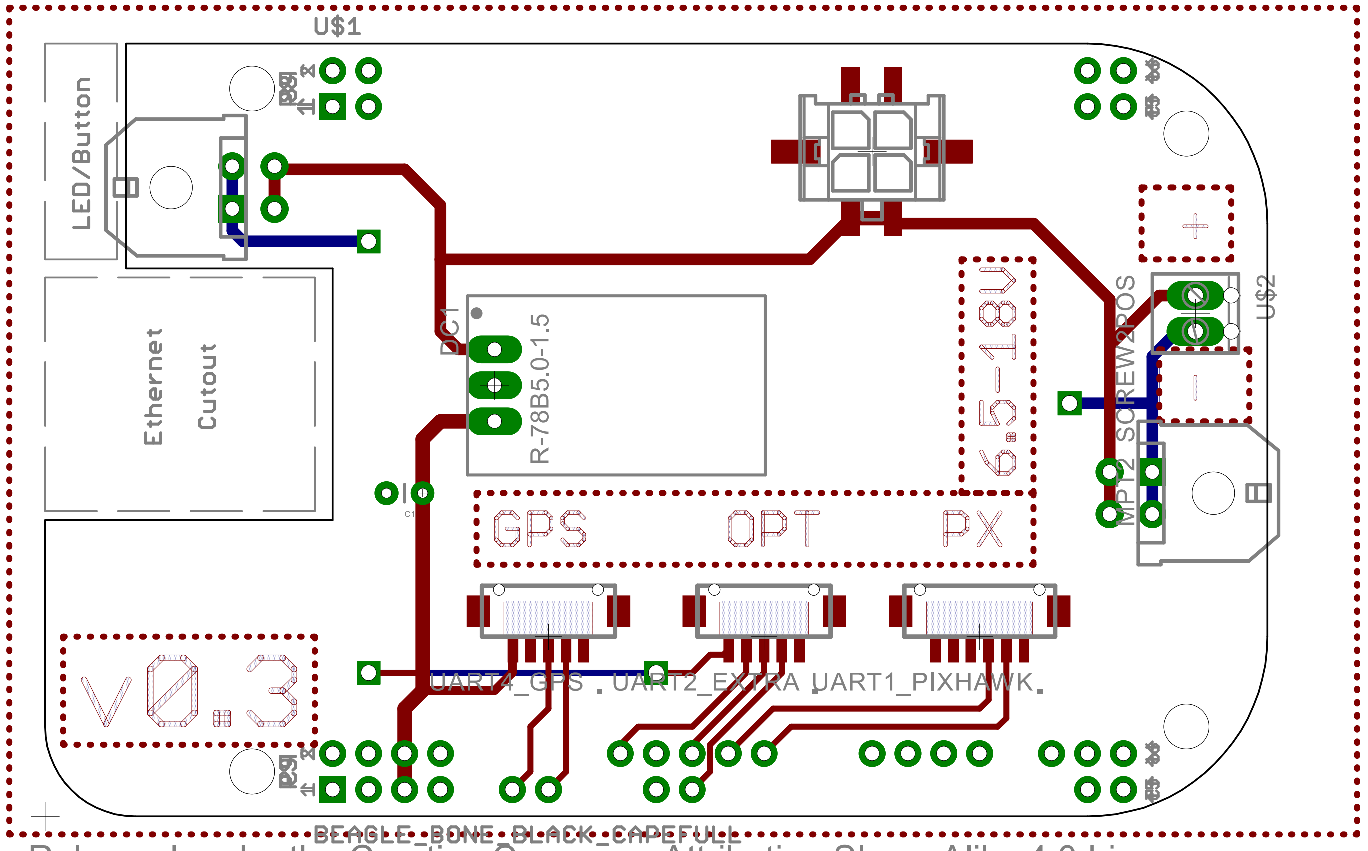
Appendix K

PCB Designs

PWM generator PCB design



PCB design for cape



Appendix L

Project Build/Compile/SIL Cheat Sheet (in Norwegian)

```
#Skifte statisk IP
sudo ifconfig eth0:1 192.168.1.67 netmask 255.255.255.0

#Lage glued med x8:006 (stå i dune/glued)
./mkconfig.bash ntnu-x8-006 && ./mksystem.bash lctr-b2xx/ntnu-x8-006.bash &&
./pktoolchain.bash lctr-b2xx/ntnu-x8-006.bash && ./pkrootfs.bash lctr-b2xx/ntnu-x8-006.bash &&
sudo ./mkdisk.bash lctr-b2xx/ntnu-x8-006.bash /dev/sde

#Lage ny task (stå i dune/dune)
python programs/scripts/dune-create-task.py ~/dune/dune/user/ "Vegard Grindheim"
PrecisionAirdrop/vegardTest

#Krysskompilere (stå i dune/bbbuild)
cmake -DCROSS=/home/vegag/dune/glued/lctr-b2xx/toolchain/bin/armv7-lsts-linux-gnueabi
../dune && make -j8; make package -j8;
så
rsync -avz /home/vegag/dune/bbbuild/dune-2.TABHER! root@192.168.1.125:/opt/lsts/dune/
så
services dune restart

#Logg på BBB
ssh root@192.168.1.125

#Start dune (stå i /home/opt/lsts/dune/)
./bin/dune -c ntnu-x8-006 -p Hardware

#eller
./dune -c x8-00 -p Simulation



#SIL TEST

#Ardupilot/pixhawk simulator (stå i ArduPlane)
./ArduPlane.elf

#Start opp simulator (stå i /home/vegag/sitl/ardupilot/Tools/autotest/jsbsim)

./runsim.py --home=-35.362938,149.165085,50,0 --script=jsbsim/rascal_test.xml

#Start MAVproxy
mavproxy.py --master tcp:127.0.0.1:5760
```

```
#Fix bug
module load map

#Send target (stå i dune/build)
./dune-sendmsg 127.0.0.1 6002 Target -35.356822 149.158661 50 3

#Send PWM
./dune-sendmsg 192.168.1.125 1025 PowerChannelControl name 1
```

Bibliography

AM3358 (2014). AM3358 | AM335x Processors | ARM Cortex-A8 Core | Description & parametrics. Available from:<http://www.ti.com/product/am3358>. [12.12.2014].

APM (2014). APM | Open source autopilot. Available from: <http://ardupilot.com/>.[12.12.2014].

APM Planner 2.0 (2014). APM Planner 2.0. Available from: <http://planner2.ardupilot.com/>.[12.12.2014].

ArduPlane Autopilot (2015). Arduplane autopilot simulator. Available from: <https://github.com/diydrones/ardupilot/tree/master/ArduPlane>.[15.05.15].

BeagleBone Black Datasheet (2014). BeagleBone Black Datasheet. Available from: <http://www.adafruit.com/datasheets/BBB_SRM.pdf>.[18.12.2014].

Beard, R. W. and McLain, T. (2012). *Small Unmanned Aircraft.* Princeton University Press.

Bencatel, R. and Girard, A. (2011). Shear Wind Estimation. (August):1–8.

Berndt, j. and Peden, T. (2015). Jsbsim. Available from: <https://github.com/tridge/jsbsim>.[15.05.15].

Brezoescu, C.-a. (2014). Small lightweight aircraft navigation in the presence of wind.

Bushuyev, A. (1970). Sea ice nomenclature. WMO. Available from: <http://www.aari.nw.ru/gdsidb/docs/wmo/nomenclature/WMO_Nomenclature_draft_version1-0.pdf>.[15.05.15].

Dias, H. (2014). LSTS » DUNE, DUNE: Unified Navigation Environment. Available from: <http://lsts.fe.up.pt/software/dune>.[12.12.2014].

Drag (2015). Drag of cylinders & cones. Available at <http://www.aerospaceweb.org/question/aerodynamics/q0231.shtml> [03.06.15].

Egeland, O. and Gravdahl, T. (June, 2003). *Modeling and Simulation for Automatic Control*. Marine Cybernetics.

Eik, K. (2010). Ice management in Arctic offshore operations and field developments.

Enge, P. K. (1994). The Global Positioning System: Signals, measurements, and performance. *International Journal of Wireless Information Networks*, 1(2):83–105.

Fortier, L. (2004). An application of a proposed airdrop planning system.

Fortuna, J. and Fossen, T. I. (Submitted). Cascaded line-of-sight path-following and sliding mode controllers for fixed-wing uavs. *2015 IEEE Multi-Conference on Systems and Control*.

Fossen, T. I. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd, Chichester, UK.

Fuglaas, S. (2014). Precision Airdrop from a Fixed-Wing Unmanned Aerial Vehicle. *Department of Engineering Cybernetics*, (June).

GLUED (2014). LSTS/glued. Available from: <https://github.com/LSTS/glued>.[12.12.2014].

Goldfein, D. L. (2013). Joint Publication 3-17 Air Mobility Operations. (September).

GPS Modules (2015). Gps modules. Available from: <https://pixhawk.org/peripherals/sensors/gps>.[15.05.15].

GPS Position Accuracy (2003). Available at <www.novatel.com/assets/Documents/Bulletins/apn029.pdf> [05.06.15].

Henry, M., Lafond, K., Noetscher, G., Patel, S., and Pinnell, G. (2010). Development of a 2,000-10,000-lb improved container delivery system. (April):0–10.

IMC Specification (2014). IMC v5.4.3-550440c - IMC v5.4.3 Specification. Available from:

<https://www.lsts.pt/imc/doc/master/>.[12.12.2014].

Johansen, T. A., Cristofaro, A., Soerensen, K., Hansen, J. M., and Fossen, T. I. (2015). On estimation of wind velocity, angle-of-attack and sideslip angle of small uavs using standard sensors. *Proc. of the 2015 International Conference on Unmanned Aircraft Systems (ICUAS'15).*

Langelaan, J., Alley, N., and Neidhoefer, J. (2011). Wind field estimation for small unmanned aerial vehicles. *Journal of Guidance, Control, ...*.

LEA-6 series u-blox (2015). *LEA-6 series u-blox 6 GPS, QZSS, GLONASS and Galileo modules.* Available from: <http://www.u-blox.com/images/downloads/Product_Docs/LEA-6_ProductSummary_%28GPS.G6-HW-09002%29.pdf>.[18.12.2014].

LSTS Toolchain (2014). Lsts toolchain for autonomous vehicles. Available from: <http://lsts.pt/neptus/info.html>.[12.12.2014].

Mathisen, S. (2014). High Precision Deployment of Wireless Sensors from Unmanned Aerial Vehicles. (May).

MAVLink Micro Air (2015). Mavlink micro air vehicle communication protocol. Available at <http://qgroundcontrol.org/mavlink/start> [03.06.15].

McGill, P., Reisenbichler, K., Etchemendy, S., Dawe, T., and Hobson, B. (2011). Aerial surveys and tagging of free-drifting icebergs using an unmanned aerial vehicle (UAV). *Deep Sea Research Part II: Topical Studies in Oceanography*, 58(11-12):1318–1326.

Nanostation M (2014). Nanostation M Wi-Fi Radio. Available from: <http://dl.ubnt.com/datasheets/nanostationm/nsm_ds_web.pdf>.[18.12.2014].

OpenWRT linux distribution (2014). OpenWRT linux distribution. Available from: <https://openwrt.org/>.[12.12.2014].

Park, S., Deyst, J., and How, J. P. (2004). A new nonlinear guidance logic for trajectory tracking. *Proceedings of the AIAA Guidance, Navigation and Control Conference*, (AIAA-2004-4900).

Parker, G. (1977). Projectile motion with air resistance quadratic in the speed. *Am. J. Phys.*

Piksi FAQ (2013). Frequently asked questions, what is the horizontal and vertical accuracy of piksi. Available at <http://docs.swift-nav.com/wiki/Frequently_Asked_Questions#What_is_the_horizontal.2Fvertical_accuracy_of_Piksi.3F> [05.06.15].

Piksi RTK-GPS (2014). Piksi RTK-GPS. Available from: <http://docs.swiftnav.com/pdfs/piksi_datasheet_v2.3.1.pdf>.[17.12.2014].

Pixhawk Autopilot (2014). Pixhawk Autopilot - PX4 Autopilot Platform. Available from: <https://pixhawk.org/modules/pixhawk>.[12.12.2014].

Plane: L1 Control (2013). Plane: L1 control for straight and curved path following. Available at <https://github.com/diydrones/ardupilot/pull/101> [03.06.15].

Puestow, T., Parsons, L., Zakharov, I., Cater, N., Bobby, P., Fuglem, M., Parr, G., Jayasiri, A., Warren, S., and Warbanski, G. (2013). OIL SPILL DETECTION AND MAPPING IN LOW VISIBILITY AND ICE: SURFACE REMOTE SENSING. (October).

R78Bxx15L Datasheet (2014). R78Bxx15L_eng_datasheet. Available from: <https://www1.elfa.se/data1/wwwroot/assets/datasheets/R78Bxx15L_eng_datasheet.pdf>.[18.12.2014].

Rocket M5 (2014). Rocket M5 Wi-Fi Radio. Available from: <http://dl.ubnt.com/rocketM5_DS.pdf>.[18.12.2014].

Servoless Payload Release (2014). Servoless Payload Release (EFLA405): E-flite - Advancing Electric Flight. Available from: <http://www.e-fliterc.com/Products/Default.aspx?ProdID=EFLA405>.[12.12.2014].

Sherwood, A. E. (1967). Effect of air drag on particles ejected during explosive cratering. *Journal of Geophysical Research*, 72(6):1783–1791.

Shestov, A. and Marchenko, A. (2014). 22 nd IAHR International Symposium on Ice. pages 660–673.

SITL Simulator (2015). Sitl simulator (software in the loop). Available at

<http://dev.ardupilot.com/wiki/simulation-2/sitl-simulator-software-in-the-loop/> [04.06.15].

Tiffin, S., Pilkington, R., and Hill, C. (2014). A Decision-Support System for Ice/Iceberg Surveillance Advisory and Management Activities in Offshore Petroleum Operations. *OTC Arctic Technology* ....

Westall, P., Carnie, R., O'Shea, P., Hrabar, S., and Walker, R. (2007). Vision-based UAV maritime search and rescue using point target detection. 2007.

Wuest, M. and Benney, R. (2005). Precision Airdrop (Largage de precision). 323(December).