NTNU – Trondheim
Norwegian University of
Science and Technology

Autonomous Inspection of Wind Turbines and Buildings using an UAV

Sondre Høglund

Master of Science in Engineering Cybernetics (2 year))
Submission date: June 2014
Supervisor: Tor Arne Johansen, ITK
Co-supervisor: Kristian Klausen, 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:	Sondre Høglund
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Autonom inspeksjon av vindturbiner og bygninger ved hjelp av en UAV
Thesis title (English):	Autonomous inspection of wind turbines and buildings using an UAV

Thesis Description: Design an observer capable of local navigation based on input from image processing algorithms such as optical flow and Hough transform. In particular, investigate how this can aid in visual inspection of wind turbines and buildings from a UAV-platform (e.g. a multicopter).

The following items should be considered:

1. Overall system description with detailed module interaction schemes and protocols.
2. Give an overview of optical flow and Hough line transform, and how its output can be used in local navigation, e.g. inspect wind turbine and building facades.
3. Investigate and set up a mathematical model of the system
4. Create an observer capable of local navigation with data from optical flow and possibly other sensors. The observer should estimate the relative distance and speed of the target object (i.e. a blade on a wind turbine) and the UAV platform.
5. Discuss how this data can be used to create a controller capable of inspecting for damages on structures, i.e. a wind turbine or building facades.
6. Create the controller capable of inspecting for damages.
7. The result should be verified by simulations. Major system components should be tested in experiments, including module communication.
8. Test the different subparts of the system (e.g. power supply, sonar).
9. Test the controller on a UAV-platform.
10. Conclude findings in a report. Include MATLAB/C-code as digital appendices together with user guide.

Start date:	2014-01-20
Due date:	2013-06-16

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

Abstract

This thesis describes the use of optical flow and Hough transform in local navigation, particularly the case of inspecting wind turbines and buildings for damages. The goal is to create an observer capable of estimating the metric velocity and distance to the blade from scaled velocity inputs. Furthermore, a controller capable of inspecting wind turbines and buildings autonomously will be presented and tested.

The first part of the report will address the use of two different computer vision algorithms in local navigation. Firstly, the optical flow algorithm will be presented, with focus on the Horn-Schunck and the Lucas-Kanade method. By using a pyramidal representation of the image, the algorithms were found to provide more accurate optical flow when the motion is large. Secondly, the Hough transform for finding straight lines in an image was investigated. The tests showed that Hough transform can be used on wind turbine blades to estimate the desired velocity vector and the relative angle between the UAV and the blade.

The observer was simulated in MATLAB with velocity vectors provided by an optical flow algorithm. Two different case studies has been investigated to verify the mathematical model and observer. The observer was able to successfully estimate the metric velocity along with the distance.

The guidance law presented in this report was based on the pure pursuit guidance and PID controllers. The controllers successfully maneuvered the UAV to the desired position and kept a constant distance to the object. The height controller was tested with both a stationary and dynamic desired height. The UAV was able to follow the desired height in both cases.

Keywords: unmanned aerial vehicle, extended Kalman filter, inertial navigation systems, inertial measurement unit, visual aided inertial navigation, object tracking, pure pursuit guidance, PID, optical flow, Lucas-Kanade method, Horn-Schunck method, Hough transform, wind turbine inspection, building inspection.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree M.Sc. in Engineering Cybernetics at the Norwegian University of Science and Technology.

I would like to thank my supervisor, Professor Tor Arne Johansen for giving me the opportunity to work on this project. I would also like to thank Ph.D. student Kristian Klausen for valuable feedback and suggestions.

Last but not least, I would like to thank my parents and Anna for their support throughout my studies.

S.H.

Problem description

The objective of this report is to design a observer and controller capable of local navigation based on input from optical flow or Hough transform. The goal is to inspect objects for damages, in particular wind turbine blades and building structures. The possible causes for damage on a wind turbine blade can be birds, lightning strike or erosion. Any damages to the blade can make the wind turbine unbalanced and at worst destroy the turbine. For this reason, it's important to discover any damages as early as possible to minimize the downtime and cost. In the case of building inspections, the structural integrity can be affected over time due to lack of maintenance and external forces such as dry weather. Any cracks or corrosion on a building can be an indicator of reduced strength in the material and needs to be investigated as soon as possible.

The motivation for using an UAV to inspect the blades and building autonomously, is to reduce the risk of injuries on operators. The location of the wind turbine and building may also make it hard and costly to inspect for humans.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

APM	ArduPilot Mega
BEC	Battery Eliminator Circuit
DOF	Degrees Of Freedom
EKF	Extended Kalman Filter
FPS	Frames Per Second
GPGPU	General-Purpose computing on Graphics Processing Units
GPS	Global Positioning System
HIL	Hardware In the Loop
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
KF	Kalman Filter
LRF	Laser Range Finder
LOS	Line-Of-Sight
MAV	Micro Air Vehicle
OS	Operating System
PDF	Probability Distribution Function
PWM	Pulse Width Modulation
SBC	Single-Board Computer
SITL	Software In The Loop
SONAR	SOund Navigation And Ranging
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
VINS	Visual Inertial Navigation System

VTOL Vertical Take-Off and Landing

Chapter 1

Introduction

The topic of this thesis is to create an observer capable of using visual and inertial information to estimate the relative position and velocity between an unmanned aerial vehicle (UAV) and an object. In addition, design a controller capable of inspecting wind turbines and buildings for damages. This chapter will start with a brief overview of the background and motivation for this thesis, and the existing solutions and recent challenges in this field of study. Furthermore, the contributions of this thesis are given, along with the objectives and limitations. Finally, the structure of the thesis is presented.

1.1 Background and motivation

The number of UAVs has increased exponentially in recent years [2, 71, 17]. The development and use of UAVs was primarily done by the military, but now the number of civilian applications have increased. The UAVs are expected to play a significant part in civilian applications such as aerial footage, search and rescue operations, power line inspections, wildlife surveillance and operate in hazardous environments were the risk of human lives are a possibility [31, 70, 38]. With these applications in mind, several papers has been written in the field of autonomous landing [63, 47, 79] and obstacle avoidance [42, 24]. Recently, EU-funded UAV projects have been created to develop service robots for use in industry [1].

The use of inertial navigation systems (INS) for estimating the position and attitude of vehicles operating in a 3D space, e.g. airplanes, helicopters and space shuttles, have been investigated for many years [56, 35]. These systems have proven to yield good local pose estimates over a short time, but bias and noise in the measurements will make the system drift. To reduce this problem, an INS can be integrated with external measurements provided by another sensor, e.g. radar, star trackers and Global Positioning System (GPS) [36, 53, 5]. This systems will exploit the complementarity in the different sensors and provide a better estimate. GPS aided INS has proven to produce accurate estimates with resolutions within a few meters, and is widely used today in both marine and aircraft applications [61, 76, 22, 62].

In situations where GPS signals are jammed or not available, i.e. under water, indoors or in urban environments, the use of cameras as an aiding sensor has been investigated. Visual aided navigation system (VINS) has become more and more popular in recent

years, and been a research topic for a long time [77]. The VINS does not rely on satellites nor any external equipment on the ground, which makes it highly robust. It also comes with the added benefit that it provides a more accurate local pose estimate. The added accuracy together with the visual information, makes the VINS able to inspect objects for damages.

1.2 Previous work

There are several approaches fusing visual information with an INS. A common approach is to use either a linear Kalman filter (KF) [46] or the nonlinear equivalent, the extended Kalman filter (EKF) [50, 34]. In [2, 49], the authors showed that the pose estimation of an UAV improved with the use of information from a camera. The visual information can also be used to land on a target [37] or to track a moving object in two [71] and three dimensions [55, 12].

Most of these papers use the cameras to estimate a global position of the UAV itself and the target it is tracking. In the field of estimating the velocity between the UAV and an object with a fixed global position, e.g. wind turbines and buildings, it is more interesting to look at the relative position and velocity. The use of optical flow for hovering and vertical landing control has also yielded good results [25]. This paper relied on a base station for doing the necessary computations and also used the non-metric scaled velocities. In [26] the authors addressed a solution to the scale issue to provide a metric velocity measurement, but with the use of a sonar. Weiss et al. presented an approach to estimate the metric scale in a world frame without the use of sonar or other sensors, but the result was used to initialize a Parallel Tracking and Mapping (PTAM) toolbox [72]. Finally, in [23] the author uses an EKF and a nonlinear observer to estimate the metric velocity measurements from the camera. However, this information was only used to compute the distance to the ground and the relative velocity between the UAV and the ground. The author did not use this information further in controllers.

1.3 Contribution and scope of this thesis

This thesis will address the use of a monocular camera together with on-board inertial sensors to inspect wind turbines and buildings for damages. Furthermore, it will estimate the relative velocity between an UAV and the target in metric velocities. Lastly, design a controller capable of inspecting the structures autonomously. The controllers will be implemented and tested on an UAV to verify the design.

1.4 Objectives

The main objectives of this project are

- **System modeling** - Investigated the different parts of the system and derive a mathematical model

- **Optical flow** - Give an overview of the optical flow algorithm and the two most used techniques. Describe how optical flow can be used in local navigation, e.g. object tracking and obstacle avoidance
- **Hough transform** - Give an overview of the Hough transform and how it can be used to detect straight lines. Describe how Hough transform can be used in local navigation, e.g. wind turbine blade inspection
- **Design of observer** - Create an observer based on the information received from optical flow and on-board inertial sensors. The observer should be able to estimate the relative distance and metric velocity of an object
- **Design of control law** - Design a controller capable of inspecting a wind turbine blade or a building with velocity vectors as input. The input should be given by the optical flow or Hough transform algorithms
- **Simulate** - Simulate the observer and controller to verify the results
- **Test** - Test the system on a UAV platform to verify the simulations

1.5 Assumptions

This thesis does not cover the stabilization of a multirotor UAV [7, 60], nor the required algorithms to achieve the optical flow or Hough transform information from one or several cameras [9, 3]. This report assumes that this is already been done, so the information from the camera and the autopilot can be used as input to the observer and controller.

1.6 Notation

The notation used in this thesis follows the notation in [18]. The representation of vectors and matrices are in bold letters, and curly brackets, {}, specifies the reference frame. Superscript is used to indicate which frame a vector is represented in, e.g. $\mathbf{v}^n$ is the velocity expressed in {n}. Relative translation of one frame with respect to another frame is written with both super- and subscript, e.g $\mathbf{v}_b^n$ is the velocity of frame b expressed in frame n.

Rotation matrices are used to perform rotations between two coordinate frames, e.g. the rotation matrix $\mathbf{R}_b^a$ is used to transform a vector from {b} to {a}

$$\mathbf{v}^a = \mathbf{R}_b^a \mathbf{v}^b$$

Rotation matrices can be combined to perform rotations between multiple frames. The rotation matrix from {c} to {a} can be expressed as a product of rotation matrix from {b} to {a} and from {c} to {b}

$$\mathbf{R}_c^a = \mathbf{R}_b^a \mathbf{R}_c^b$$

Throughout this thesis, the identity matrix, $\mathbf{I}$, and zero matrix, $\mathbf{0}$, are frequently used. The subscripts indicates the size of the matrices, e.g. $\mathbf{I}_4$ is the 4×4 identity matrix and

$\mathbf{0}_{2\times4}$ is a 2×4 matrix filled with zeros.

For a vector $\boldsymbol{\lambda} = [\lambda_1 \quad \lambda_2 \quad \lambda_3]^T$, $\mathbf{S}(\boldsymbol{\lambda})$ describes the 3×3 skew-symmetric matrix for a cross product

$$\boldsymbol{\lambda} \times \mathbf{a} := \mathbf{S}(\boldsymbol{\lambda})\mathbf{a}$$

where $\mathbf{S} \in SS(3)$ is

$$\mathbf{S}(\boldsymbol{\lambda}) = -\mathbf{S}^T(\boldsymbol{\lambda}) = \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix}$$

1.7 Structure of the thesis

The rest of the thesis is structured as follows. Chapter 2 describes the different components needed to inspect the wind turbine blade and building presented in this project. In Chapter 3, the necessary equations for the system under consideration are presented. Chapter 4 will give a brief overview of the computer vision used in this thesis and how this information can be used in local navigation. These equations are then used to design an observer in Chapter 5. Afterwards, in Chapter 6, the design of a guidance law and controller is presented. The system is then simulated in Chapter 7 to verify the model and observer. Furthermore, in Chapter 8 the system is simulated with the necessary software in the loop, thus verifying that the controllers are applicable to the UAV. In Chapter 9 the different parts are tested on a UAV platform. Finally, the results are concluded together with an overview of the future plans in Chapter 10.

Chapter 2

System description

This chapter presents an overview of the different parts of the system used in this thesis. This includes the necessary hardware, e.g. the UAV, camera, autopilot, on-board computer and distance sensor. A description of the wind turbines and buildings considered in this thesis is also given. Lastly, the coordinate frames to the respective components are presented together with the necessary software.

2.1 UAV

The UAV used in this thesis needs to be a vertical take-off and landing (VTOL) vehicle and be able to move in 6 degrees of freedom (6DOF), e.g. a multicopter. A multicopter is a rotary-wing aircraft consisting of more than two rotors. The rotors can be controlled independently to stabilize, provide thrust and control the aircraft. In the case of a hexacopter, a multicopter with 6 motors, three of the propellers rotate clockwise while the rest rotate counter clockwise, see Figure 2.1. This setup will cancel out the rotational contribution about the z-axis. The multicopter is controlled using an inertial measurement unit (IMU) strapped down to the body.

In this thesis a 3D Robotics hexacopter has been used. The specifications are available in Table 2.1.

2.2 Camera

To observe the relative motion and compute the optical flow and Hough transform, a camera is needed. The camera is mounted to the UAV with the optical sensor facing forward. This thesis considers a pinhole camera, see Figure 2.2. There are several features to take into account when deciding on a camera, e.g. frames per second (FPS), resolution and the focus length. The choice of these features depends on the desired capabilities of the camera and the computational power of the on-board computer.

2.3 Autopilot

The autopilot used in this project is the open source, Arduino based ArduPilot Mega (APM). The APM stabilizes the UAV with use of a 6DOF IMU, consisting of a 3DOF

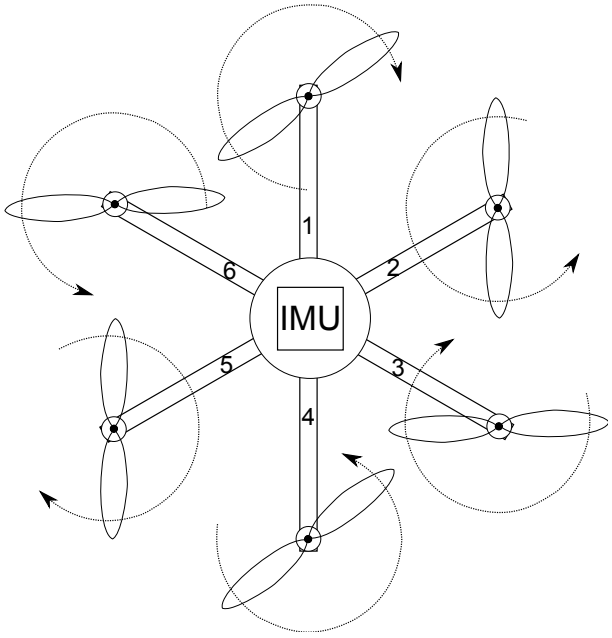
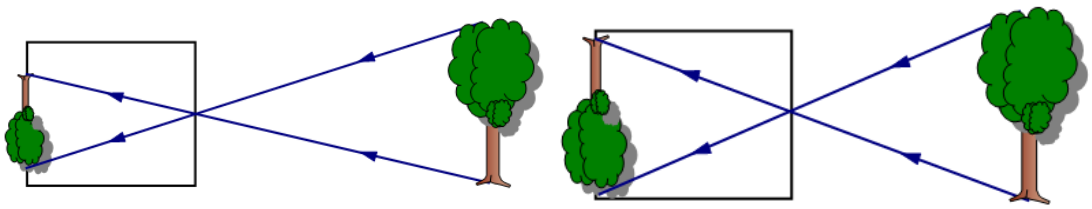


Figure 2.1: Principle drawing of a hexacopter. Motor 1, 3 and 5 rotates clockwise, while motor 2, 4 and 6 rotates counter clockwise. The IMU is located in the center of gravity.

Table 2.1: Specifications hexacopter

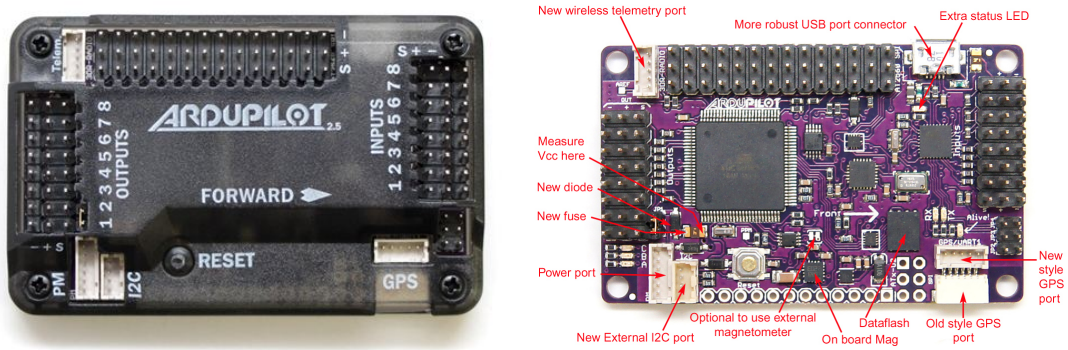
Components	**Detailed description**
Frame	3DR hexacopter frame with custom 3D printed payload
Motors	880Kv AC2836-358
ESC	20A ESC with SimonK firmware
Autopilot	APM 2.6 ArduPilot
USB Low quality camera	Econ e-CAM51_USB
HQ Camera	GoPro Hero3+ Black Edition
On-board computer	Pandaboard ES
GPS	3DR GPS uBlox LEA-6 Board
Power module	APM Power Module with XT60 Connectors
Telemetry	3DR Telemtry Kit - 433 MHz (Europe)
RC controller	Spectrum DX7s 7 CH Transmitter with AR8000
Battery	Hyperion 3s 4000mAh 25C



(a) Camera far away from object

(b) Camera closer to object

Figure 2.2: Principle drawing of a pinhole camera



(a) with enclosure

(b) without enclosure

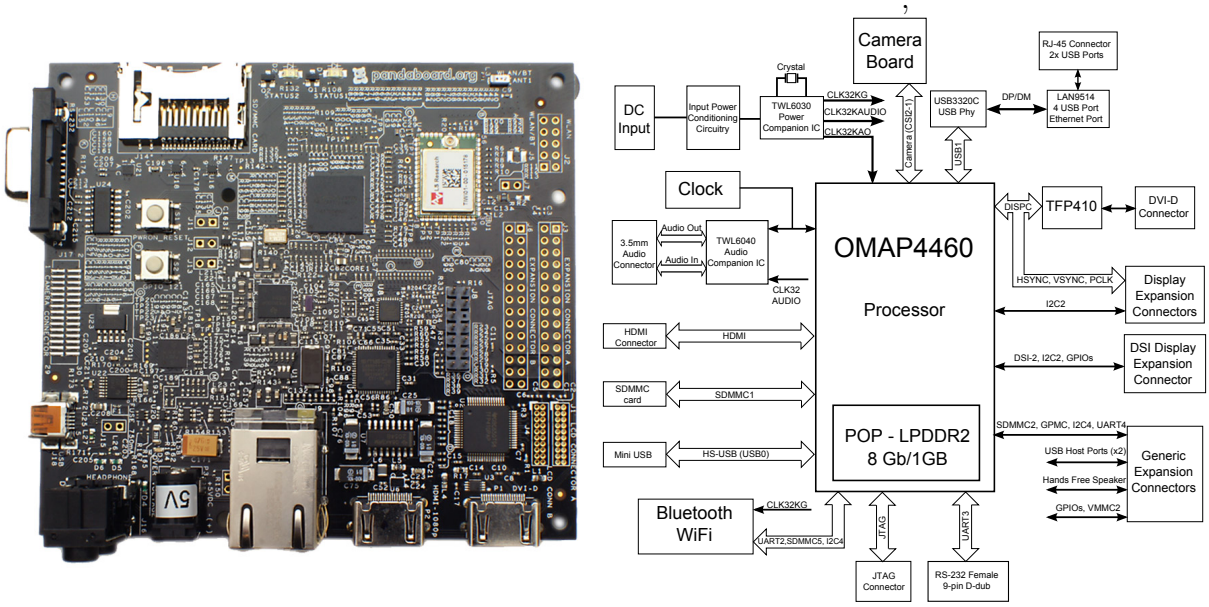
Figure 2.3: Picture of an APM with and without enclosure. The output pins is to connect the different motors, while the input pins are to connect a receiver. There are also port for telemetry, GPS and external I2C sensors.

gyroscope and a 3DOF accelerometer. In addition, the APM has a magnetometer and a barometer to measure the heading and altitude of the UAV. It is also possible to connect optional external sensors, e.g. GPS, pitot tube and telemetry, via Inter-Integrated Circuit (I2C) or a serial port. When a GPS is connected, the UAV has the opportunity to return home when it is out of sight or loses radio control. The pitot tube is used to measure the wind speed, thus calculated the relative speed of the UAV. Telemetry is a two-way communication link between the UAV and a ground station running Mission Planner. In the Mission Planner software, it is possible to set waypoints, tune the parameters in-air and display on-board video if a camera link is provided. There is also a 4MB onboard memory capable of logging the mission automatically during flight.

The Mission Planner can also be used as a hardware-in-the-loop (HIL) simulator. HIL simulation makes it possible to simulate the behavior of the multicopter without the complexity and risk of flying outdoors. For this reason, new algorithms and software can be tested without any consequences. The HIL simulation can either be done in FlightGear or X-plane 9/10.

2.4 PandaBoard

Now that the different components needed to stabilize the UAV and detecting an object are established, a computer to connect the different modules and interact between them is necessary. In this thesis this is accomplished using a single-board computer (SBC). A SBC has all the necessary components on a single circuit board. This includes a central processing unit (CPU), a graphics processing unit (GPU), random-access memory (RAM) and the input-output (I/O) needed to communicate with the board. The requirements for the SBC considered in this thesis is low weight, low power consumption and high computational power. Frederik Stendahl Leira investigated different SBCs available on the market and made a decision [40]. The choice fell on the PandaBoard ES and will be used in the rest of this thesis. A top view of the PandaBoard together with a block diagram is available in Figure 2.4. Besides having presoldered components, e.g. USB, HDMI



(a) Top view (b) Block diagram

Figure 2.4: Pandaboard. Top view (a) and the block diagram (b) describing the interactions between different components

and ethernet, the PandBoard ES also offer different expansion ports to connect external sensors and equipment, e.g. I2C sensors, GPS, and camera. The full specifications are listed in Table 2.2

In order to communicate with the different hardware components, an operating system (OS) is required. PandaBoard ES supports various Linux-based operating system, e.g. Android, Firefox OS and an optimized version of Ubuntu. Since the PandaBoard does not include NAND memory, the operating system needs to be installed on an external SD memory card.

The integrated SGX540 GPU opens up the possibility of GPU programming, also known as general-purpose computing on graphics processing units (GPGPU). GPGPU programming is to utilize the GPU to perform numerical calculation usually done by the CPU. The GPU offers hundreds of cores together with high speed memory to massively parallel the computation. Prior to OpenCV v.2.4.3, GPU programming was only available using CUDA on NVIDIA GPUs. As of v.2.4.3, OpenCV has included an OpenCL module which makes it possible to program on GPUs from multiple vendors, e.g. Apple, Intel, AMD and ARM. The available GPU accelerated functions which are relevant for this project are; corner detectors, GoodFeaturesToTrack and optical flow. For a full list of all the available GPU accelerated functions in OpenCV-CL v.2.4.6 and up, see [20].

2.5 Power supply

In order for the PandaBoard to do the calculations and operations, a power supply is required. The electric speed controller (ESC) features a built-in battery eliminator circuit

Table 2.2: PandaBoard ES specification. Courtesy of [54]

Components	**PandaBoard ES**
CPU	Dual-core 1.2 GHz ARM® Cortex™-A9
GPU	Imagination Technologies' POWERVR™ SGX540
RAM	1 GB low power DDR2
Connectivity	
Wi-Fi	802.11 b/g/n (based on WiLink™ 6.0)
Bluetooth	v2.1 + EDR (based on WiLink™ 6.0)
Ethernet	Onboard 10/100
Display and Audio output	
HDMI	v1.3 Connector (Type A)
DVI-D	Can drive 2nd display
Audio in/out	3.5mm jack and HDMI (output only)
Expansion	
USB OTG	1x USB 2.0 on-the-go port, 2x USB host ports
General purpose expansion header	I2C, USB, MMC
Camera	Camera expansion header
LCD	LCD signal expansion using a single set of resistor banks
Dimensions	
Height	4.5" (114.3mm)
Width	4.0" (101.6mm)
Weight	2.6 oz (81.5 gram)

(BEC), but it is only designed to deliver power to the RC receiver and servos. Thus, it can not deliver enough current to power the PandaBoard. Instead, an universal BEC (UBEC) will be used. An UBEC is a swithcing mode power supply, as opposed to the linear power supply found in a BEC. Consequently, the UBEC can deliver more power with a significant lower weight and size. In addition, the heat in an ESC with a built-in BEC, generated by the current drawn by the motors, can cause loss of power to the PandaBoard. Worst case scenario, this can result in a crash of the UAV. By using an UBEC with an external battery, the current drawn by the UAV will not affect the PandaBoard. Likewise, a failure on the PandaBoard will not cause the UAV to lose power and crash.

2.6 Distance sensor

To be able to measure the distance to the wind turbine blade, a distance sensor is needed. It is important to know the distance to the blade, so that the hexacopter avoids crashing into the blade and risking damage to the hexacopter or blade. There are several sensor with different techniques used to measure distance, e.g. ultrasonic, laser, radar and methods including trigonometry. Due to weight consideration, availability and price, ultrasonic sonar and laser will be investigated further in this section.

2.6.1 Sonar

Sound navigation and range (SONAR) uses sound propagation to estimate the distance to an object. This is the same techniques bats uses to navigate in caves and communicate. The sonar transmits sound waves and measures the time it takes for the sound waves to reflect back, see Figure 2.5. Hence the distance can be described by the following equation

$$D = \frac{c_a t}{2} \tag{2.1}$$

where c_a is the speed of sound in air and t is the time the sound wave uses to the object and back. A sonar can also be used under water by changing the speed of sound in air to water, c_w.

The sonar systems uses a wide spectre of frequencies, from infrasonic[1] to ultrasonic[2]. Sonar was in the beginning design for, and used by, the military to either communicate with other vessels or guide submarines in navigation. In recent years, due to reduced size and cost, sonar has been more and more popular in robotic applications.

When choosing whether or not a sonar is applicable, it is important to consider the positive and negative aspects of a sonar. First of all, a sonar is low cost, light and uses very little power. It is also possible to use multiple interfaces to connect to a sonar, e.g. serial, pulse width modulation (PWM) and analog. On the other hand, a sonar is sensitive to electric noise, turbulence and reflections from surrounding obstacles. A sonar also send the sound waves at a specific cone-shaped signal, which may cause misleading measurements, see Figure 2.7. Lastly, the sonars used in robot applications are not able

[1]Sound frequency lower than 20 Hz
[2]Sound frequency greater than 20 kHz

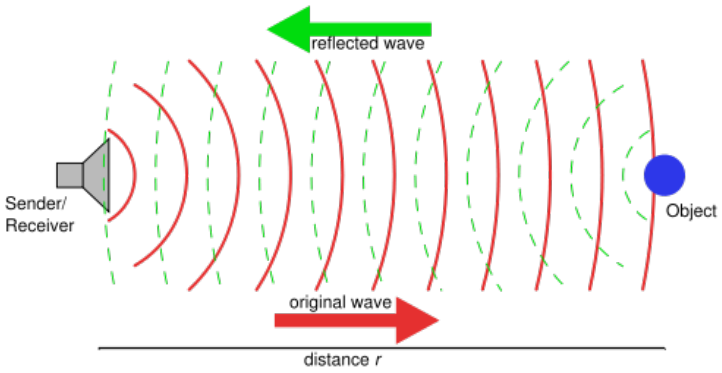


Figure 2.5: Principle drawing of a sonar. Courtesy of [68]

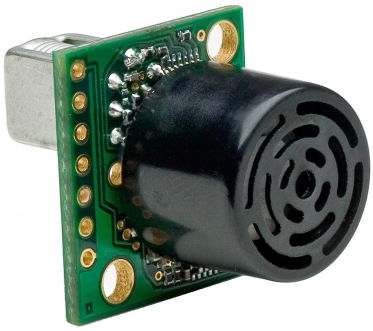


Figure 2.6: MaxBotix MaxSonar EZ4. Courtesy of [44]

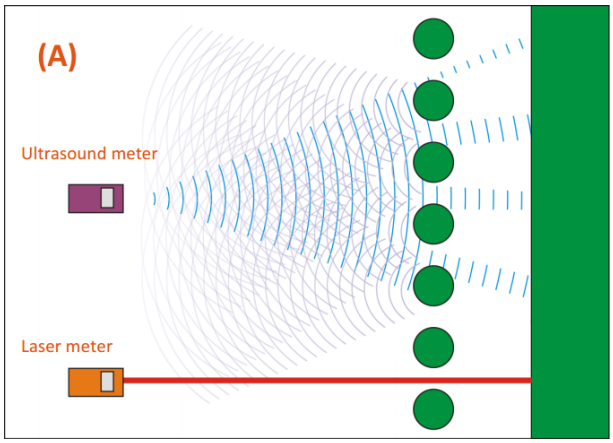


Figure 2.7: Comparison of sonar and laser measuring distance between obstacles. Courtesy of [30]

to measure objects at great distances ($>$ 10 metres).

The sonar used in this project is the MaxBotix MaxSonar EZ4, see Figure 2.6 and Table 2.3. The MaxSonar EZ4 is supported by the ArduPilot out-of-the-box and its application is usually to estimate the height above ground to make the take-off and landing autonomous. In this case, reflections and turbulence is not an issue. Regarding estimating the distance to a wind turbine, the sonar may experience reflections from the hexacopter legs and turbulence from the propellers. This will be investigated further in section 9.1 and 9.3.1.

2.6.2 Laser

While a sonar uses sound wave to estimate the distance, a laser range finder (LRF) uses the speed of light. A LRF emits a laser beam with known wavelength and measures the number of cycles. Similarly as the sonar, the distance between the LRF and an object is described as

$$D = \frac{c_l t}{2} \tag{2.2}$$

Table 2.3: MaxSonar EZ4 MB1240 specification. Courtesy of [44]

Connections	**Maxsonar EZ4**
Operational voltage	3.3V - 5V
Interface	Analog, pulse width and serial.
Features	
Update frequency	10Hz (Readings can occur every 100ms)
Range	0 - 765cm
Resolution	1cm
Dimensions	
Height	0.78” (19.9mm)
Width	0.870” (22.1mm)
Depth	0.989” (25.11mm)
Weight	0.208 oz (5.9 gram)

where c_l is the speed of light. The time it takes for a wave to hit the object and reflect back is too fast to measure precisely. Hence, another method for measuring the time is needed. The time can be calculated as [57]

$$t = \frac{\phi}{\omega} \tag{2.3}$$

where ϕ is the phase shift and ω is the angular frequency of the light beam. Substituting (2.3) in (2.2) yields

$$D = \frac{1}{2}\frac{c\phi}{\omega} = \frac{c}{4\pi f}(N\pi + \Delta\phi) = \frac{\lambda}{4}(N + \Delta N)$$

where N is the whole number of wavelengths, $\Delta N = \Delta\phi/\pi$ is the residual of the wave and $\lambda = c/f$ is the wavelength of the laser. From the equation we see that the distance is calculated as a function of the wavelength of the laser and number of cycles.

As opposed to an ultrasonic sonar, a LRF can measure considerable distances[3]. In addition, a LRF estimates the distance using a single point and will not be affected by reflections or turbulence. The disadvantages of using a LRF is the higher weight, price and update frequency.

The LRF considered in this thesis is the LightWare SF02, see Figure 2.9 and Table 2.4. The SF02 is specifically designed for use on UAVs with a range of 40 metres and 1cm resolution. The module can be connected to the PandBoard either through the analog pins or by using serial communication.

2.6.3 Discussion

This section has shown that the sonar and the laser module have the same resolution. When it comes to range, power consumption and size there are noticeable differences. The SF02 module has longer range and will not be affected by reflections and turbulence.

[3]Some handheld LRF can measure hundreds of metres

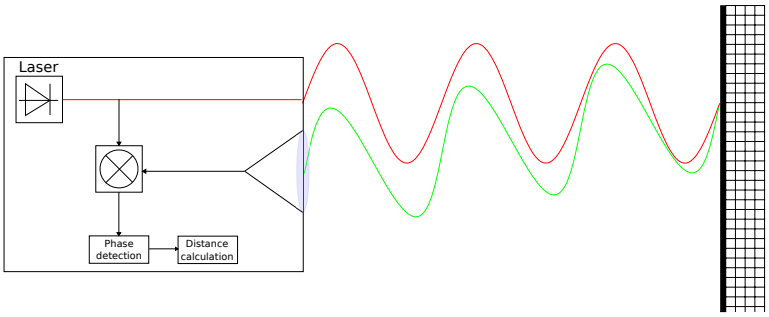


Figure 2.8: Principle drawing of a LRF. The clear red wave is the emitted light and the green is the return wave. Based on [10]

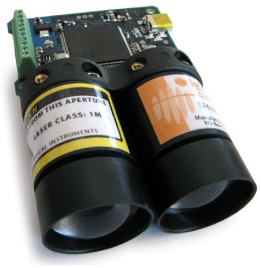


Figure 2.9: LightWare SF02. Courtesy of [52]

Table 2.4: LightWare SF02 specification. Courtesy of [52]

Connections	**SF02**
Operational voltage	6.5-9V unregulated or 5V regulated. Also possible to connect both unregulated and regulated power for redundancy.
Interface	Analog and serial.
Features	
Update frequency	12Hz
Range	0 - 40m
Resolution	1cm
Dimensions	
Length	86mm
Width	59mm
Height	27mm
Weight	75 gram

Figure 2.10: Modern wind turbine

(a) Lightning strike (b) Erosion

Figure 2.11: Two examples of damage of a wind turbine blade. Courtesy of [39]

However, it uses more power, is larger and over ten times heavier. The EZ4 sonar has shorter range, but with a range of over 7 metres it is far enough in the wind turbine and building inspection scenario.

2.7 Applications

This thesis will focus on two different inspection applications, wind turbine blades and buildings. In this section, the two different objects will be presented and the different types of damage will be discussed.

2.7.1 Wind turbine

The wind turbine considered in this thesis is based on the three bladed modern wind turbine, see Figure 2.10. Although the wind turbine in the picture is located on land, the methods and results presented in this thesis are also applicable to wind turbines offshore. As previously mentioned, damage on the blades can in a worst case scenario destroy the wind turbine. Consequently, it is important to discover any damages as early as possible to minimize the cost and maximize the production of electricity. Two of the most common damages, lighting strike and erosion, are shown in Figure 2.11. Wind turbines are commonly found along the coastline to utilize the wind. However, this location makes the wind turbines vulnerable to collisions with large birds of prey.

2.7.2 Building

The other application consider in this thesis, is inspections of buildings for damage. Damage can occur due to natural disasters such as hurricanes and earthquakes, as well as weather changes. Changes in humidity or temperature can cause concrete buildings to develop cracks and deep gaps. It is important to detect the cracks as soon as possible, to prevent further development. Worst case scenario, the damages can be a sign of foundational failure.

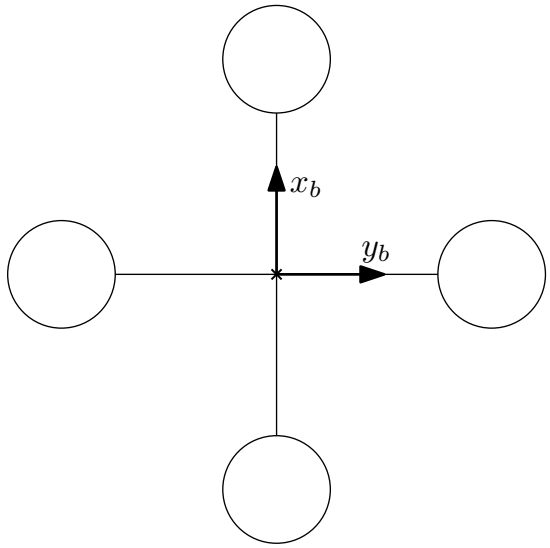
2.8 Reference frames

In order to represent the relative position between the UAV and an object, the different coordinate frames need to be defined. The different coordinate frames are explained in detail below and in Figure 2.12

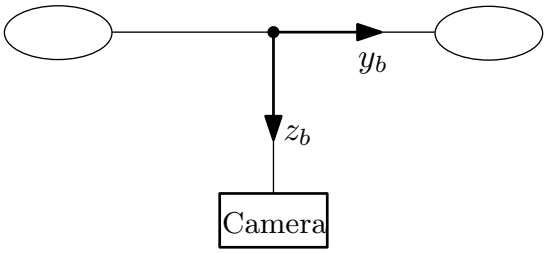
- BODY {b} - The body-fixed reference frame is a moving coordinate frame fixed to the UAV with origin at the center of gravity (CG). The linear accelerations and angular velocities are measured in this frame. The x-axis points in the direction of forward motion, while the y-axis points to the right. The z-axis points down to complete orthogonal coordinate system.

- NED {n} - The North-East-Down (NED) coordinate frame is tangential with the surface of the earth and moves with the UAV. This thesis only considers UAVs that move at low speed, so the earth's rotation can be neglected. This means that the NED frame is seen as an inertial frame in which Newton's laws of motion applies. In this frame the x-axis points towards true north, y-axis points towards east and to complete the orthogonal coordinate system, the z-axis points down.

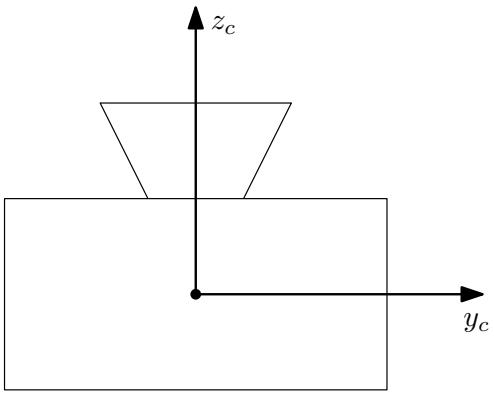
- CAMERA {c} - The camera-fixed frame is a moving coordinate frame fixed to the camera with its z-axis expanding positive from the camera to the image. The y-axis coincides with the body's y-axis. The x-axis points up to complete the orthogonal coordinate system.



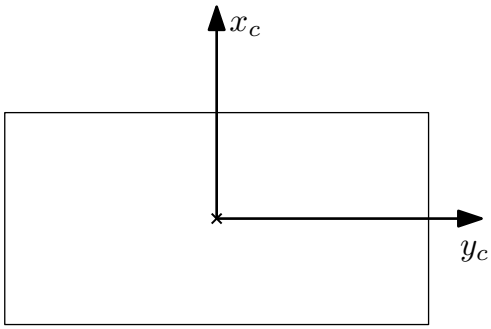
(a) Top view of the quadcopter. The z-axis points into the xy-plane



(b) Back view QC. The x-axis points out of the yz-plane



(c) Top view camera. The x-axis points out of the yz-plane



(d) Back view camera. The z-axis points into the xy-plane

Figure 2.12: Principal sketch of the UAV and camera with coordinate frames

2.9 Software

The softwares running on-board the UAV and the ground station are DUNE and Neptus, respectively. The software is developed by The Underwater Systems and Technology Laboratory (LSTS) in Porto.

2.9.1 DUNE

DUNE: Unified Navigational Environment is the system running on-board the hexacopter. DUNE handles the interaction between sensors and actuators, and the code used in control and navigation of the UAV. DUNE works by creating different separate tasks running in indepentent threads. Each task is responsible for its own function, e.g. read sensor or send commands. DUNE is operatating system (OS) and CPU architecture independent. This means that a change in OS and/or architecture of payload will not require any change of code in DUNE. Hence, the UAV will be highly portable and a change in hardware will not create additional work. In the DUNE source code, the necassary drivers for a large number of sensors are included. These are sensors that are related to navigation and control of a hexacopter or other unmanned vehicles. In addion, DUNE features several built-in algorithms such as Kalman filter and other navigation filter.

2.9.2 Neptus

Neptus is a ground control station (GCS) for commanding and communicating with one or several unmanned vehicles. In Neptus, an operator can control and send commands to individual vehicles. Neptus can be used in different stages of the mission. Firstly, Neptus can plan missions ahead time and simulate to validate the different parts of the mission, e.g. battery life, sensors and maneuvers. Secondly, Neptus can be used to receive and visualize information about several types of unmanned vehicles during the mission and give maneuver commands to the vehicle. An example of this is to set waypoints, load plans or command desired speed. Lastly, it can be used to review and analyse previous missions. The data collected during the mission can be examined in detailed. This is useful to improve the behavior of the vehicles and do the necessary corrections to the next mission.

2.9.3 IMC

Inter-Module Communication (IMC) is a protocol used for communicating between vehicles, sensors and operator consoles. The IMC protocol is used by both DUNE and Neptus to communicate and send commands. IMC also serves as the inter-process communication in DUNE to allow different tasks to run independently and communicate. The IMC protocol, in cooperation with DUNE and Neptus, has been comprehensively tested by LSTS, with successful results [58]. Figure 2.13 shows how IMC connects the vehicles running DUNE and ground stations running Neptus.

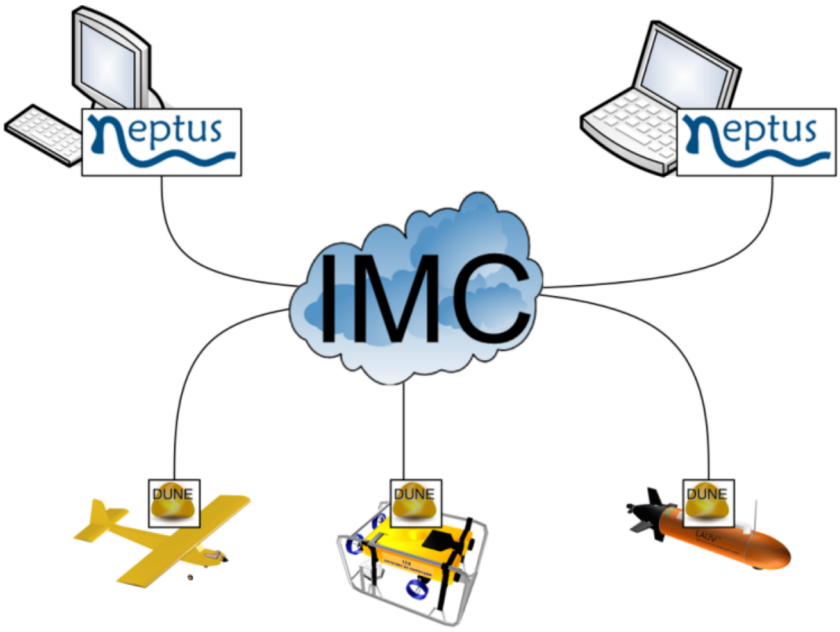


Figure 2.13: Software tool chain. Courtesy of [59]

Chapter 3

System modeling

In order to estimate the metric velocity and the distance to an object, the equations of motion are required. The following chapter addresses the derivation of these equations using standard kinematics.

3.1 Acceleration

The acceleration of a rigid body can be written as [14]

$$\vec{a}_p = \frac{{}^b d}{dt}\vec{v}_o + \vec{\omega}_{ib} \times \vec{v}_o + \frac{{}^b d^2}{dt^2}\vec{r} + 2\vec{\omega}_{ib} \times \frac{{}^b d}{dt}\vec{r} + \vec{\alpha}_{ib} \times \vec{r} + \vec{\omega}_{ib} \times (\vec{\omega}_{ib} \times \vec{r}) \tag{3.1}$$

where $\vec{r}$ is the vector between the IMU and camera. Since the camera is fixed on the body, $\vec{r}$ is constant in {b}. Hence, there is no Coriolis acceleration and (3.1) can be written

$$\vec{a}_p = \frac{{}^b d}{dt}\vec{v}_o + \vec{\omega}_{ib} \times \vec{v}_o + \vec{\alpha}_{ib} \times \vec{r} + \vec{\omega}_{ib} \times (\vec{\omega}_{ib} \times \vec{r}) \tag{3.2}$$

Writing (3.2) in coordinate form yields

$$\begin{aligned} \dot{\mathbf{v}}^c &= \mathbf{R}_b^c \left(\mathbf{a}^b + \mathbf{S}(\boldsymbol{\omega}^b)\mathbf{v}^b + \mathbf{S}(\dot{\boldsymbol{\omega}}^b)\mathbf{r}^b + \mathbf{S}^2(\boldsymbol{\omega}^b)\mathbf{r}^b \right) \\ &= \mathbf{R}_b^c \left(\mathbf{a}^b + \mathbf{S}(\dot{\boldsymbol{\omega}}^b)\mathbf{r}^b + \mathbf{S}^2(\boldsymbol{\omega}^b)\mathbf{r}^b \right) + \mathbf{S}(\boldsymbol{\omega}^c)\mathbf{v}^c \end{aligned} \tag{3.3}$$

where $\dot{\mathbf{v}}^c$ is the acceleration of the UAV expressed in {c} and $\boldsymbol{\omega}^c = \mathbf{R}_b^c \boldsymbol{\omega}^b$. The rotation matrix $\mathbf{R}_b^c$ can either be found manually by measure the angles between the camera and body frame, ϕ_c, θ_c, ψ_c, or by using a self-calibration algorithm [48, 33]. The self-calibrating algorithm is beyond the scope of this thesis, thus the rotation matrix is calculated as a combination of simple rotation around ψ_c, θ_c, and ϕ_c respectively.

$$\mathbf{R}_b^c = \mathbf{R}_{z,\psi_c}\mathbf{R}_{y,\theta_c}\mathbf{R}_{x,\phi_c}$$

The linear acceleration and angular velocity measured by the IMU is denoted by $\boldsymbol{\omega}^b$ and $\mathbf{a}^b$, respectively. The IMU will not only measure the acceleration caused by the UAV, but also the acceleration due to gravity. Hence

$$\begin{aligned} \mathbf{a}^b &= \mathbf{f}^b + \mathbf{g}^b \\ &= \mathbf{f}^b + \mathbf{R}_n^b \mathbf{g}^n \end{aligned} \tag{3.4}$$

with the rotation matrix given by

$$\mathbf{R}_n^b = \begin{bmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ s\phi s\theta c\psi - c\phi s\psi & s\phi s\theta s\psi + c\phi c\psi & s\phi c\theta \\ c\phi s\theta c\psi + s\phi s\psi & c\phi s\theta s\psi - s\phi c\psi & c\phi c\theta \end{bmatrix}$$

where $\mathbf{g}^n = [0\ 0\ g]^T$ is the gravitational vector in $\{\mathrm{n}\}$, $s\cdot = sin(\cdot)$ and $c\cdot = cos(\cdot)$. Since the IMU only measure linear acceleration and angular velocity, there is no direct measurement of the angular acceleration, $\dot{\boldsymbol{\omega}}$. Instead of performing a numerical differentiation, the assumption $\dot{\boldsymbol{\omega}} \approx 0$ is made. Using this approximation together with (3.4), we can rewrite (3.3)

$$\dot{\mathbf{v}}^c \approx \mathbf{R}_b^c \left(\mathbf{f}^b + \mathbf{R}_n^b \mathbf{g}^n + \mathbf{S}^2(\boldsymbol{\omega}^b)\mathbf{p}_{bc}^b \right) + \mathbf{S}(\boldsymbol{\omega}^c)\mathbf{v}^c \tag{3.5}$$

3.2 Distance

The distance, d, from the image plane to the origin of $\{\mathrm{c}\}$ can be written using the Hessian normal form [73]

$$\mathbf{n}^c \cdot \mathbf{P}^c + d = 0$$

where $\mathbf{n}^c$ is the plane unit normal vector and $\mathbf{P}^c$ is the principal point, i.e. the intersection of the optical axis and the image plane. The derivative of the distance can be found according to [21]

$$\dot{d} = \mathbf{n}^c \mathbf{v}^c \tag{3.6}$$

Figure 3.1 shows a graphical interpretation of the Hessian normal form. As the figure clearly shows, the plane unit normal vector can be expressed in $\{\mathrm{c}\}$ as $\mathbf{n}^c = [0\ \ 0\ \text{-}1]^T$

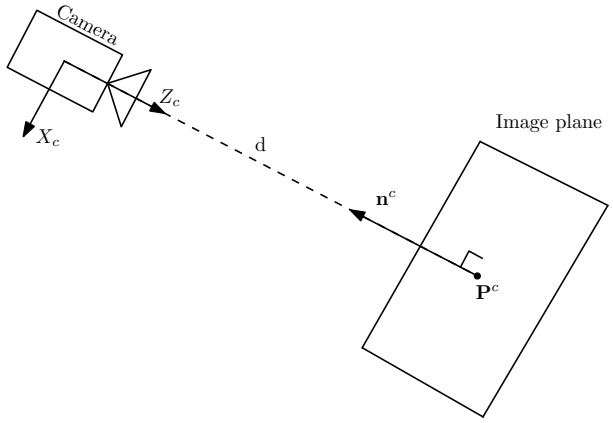


Figure 3.1: The camera projects the 3D object onto a 2D image plane. The plane unit normal vector is in the opposite direction of the camera optical axis

Chapter 4

Computer vision

This chapter describes two different computer vision algorithms. The first is the feature-based optical flow and the second is the straight line Hough transform. Both of these approaches will be examined and the applications in local navigation will be discussed.

4.1 Optical flow

This section gives an overview of the optical flow and states the two most popular methods used today. The first method is the much celebrated Horn-Schunck method and the second is the least-square based Lucas-Kanade method. The Lucas-Kanade method will receive the most focus due to its popularity and best overall performance [19]. Finally, a discussion of how the the optical flow output can be used in local navigation is presented.

4.1.1 Overview

The motion in an image is the real motion of a 3D object projected onto a 2D plane. Optical flow is the relative motion between an observer (e.g. camera, eye) and the surrounding objects in the field of view. It has been shown that insects use optical flow to navigate and avoid obstacles [67]. Figure 4.1 shows a plan view of a fly using optical flow to detect and avoid an obstacle. The same principle can be used by robots to either avoid obstacles or track objects using a camera. In Figure 4.2, a Rubik's cube is rotated on a turntable and the resulting optical flow is calculated. The figure shows that it is not possible to calculate optical flow on the turntable itself due to lack of features to track.

4.1.2 Calculation

Let $E(x, y, t)$ be the image brightness in a point (x, y) at time t. In optical flow, the algorithms assume that the brightness of a particular point in the image is conserved between frames, i.e.

$$\frac{dE(x,y,t)}{dt} = 0$$
$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

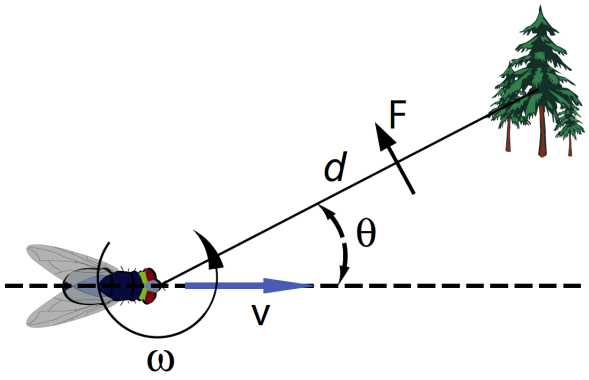


Figure 4.1: The optical flow, F, experienced by a fly travling past an obstacle. Courtesy of [29]

(a) First frame (b) Second frame (c) Optical flow

Figure 4.2: Optical flow given by a rotating Rubik's cube. Courtesy of [16]

Defining $u = dx/dt$, $v = dy/dt$ and E_x, E_y, E_t to be the partial derivatives yields

$$\begin{aligned} E_x u + E_y v + E_t &= 0 \\ \nabla \mathbf{E} \cdot \mathbf{v} + E_t &= 0 \end{aligned} \tag{4.1}$$

where $\mathbf{v} = \begin{bmatrix} u & v \end{bmatrix}^T$ and $\nabla \mathbf{E} = \begin{bmatrix} E_x & E_y \end{bmatrix}$. This equation is referred to as the gradient constraint equation [4]. The equation is linear with two unknowns for each pixel, $\mathbf{v} = \begin{bmatrix} u & v \end{bmatrix}^T$, and is known as the aperture problem in the optical flow algorithms [69]. The number of unknowns will increase proportional with the number of pixel. Hence, additional constraints are necessary in order to solve (4.1).

Horn-Schunck method

Horn and Schunck [27] combined (4.1) with a global smoothness constraint to estimate $\mathbf{v}$ by minimizing

$$\iint \left(\lambda^2 (\| \nabla u \|_2^2 + \| \nabla v \|_2^2) + (\nabla \mathbf{E} \cdot \mathbf{v} + E_t)^2 \right) dxdy \tag{4.2}$$

Here, λ is a weighting factor and chosen as the expected noise in the estimates of $E_x^2 + E_y^2$. In order to find the velocity that minimize (4.2), an iterative equation is used [65]

$$u = \bar{u} - E_x \frac{E_x\bar{u} + E_y\bar{v} + E_t}{\lambda^2 + E_x^2 + E_y^2}$$
$$v = \bar{v} - E_y \frac{E_x\bar{u} + E_y\bar{v} + E_t}{\lambda^2 + E_x^2 + E_y^2}$$

where $\bar{u}^n$ and $\bar{v}^n$ is the average of the velocity estimates u^n and v^n.

Lucas-Kanade method

Lucas and Kanade proposed an alternative solution to the aperture problem by using least-squares fit [41]. The Lucas-Kanade method considers a small neighbourhood, called a window, around the pixel and assumes that the optical flow of each pixel within this window is the same. This is a consequence of the smoothness assumption. The size of the window can be chosen arbitrary. With a window size of 4×4, the optical flow equation (4.1) can be calculated in each of the 16 pixel [65]

$$\begin{aligned} E_{x1}u + E_{y1}v &= -E_{t1} \\ E_{x2}u + E_{y2}v &= -E_{t2} \\ &\vdots \\ E_{x16}u + E_{y16}v &= -E_{t16} \end{aligned}$$

The equations above can be organized and stacked to form a linear system

$$\begin{bmatrix} E_{x1} & E_{y1} \\ E_{x2} & E_{y2} \\ \vdots & \vdots \\ E_{x16} & E_{y16} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} E_{t1} \\ E_{t2} \\ \vdots \\ E_{t16} \end{bmatrix}$$

By writing the above equations on matrix form, the optical flow can be calculated as

$$\begin{aligned} \mathbf{A}\mathbf{v} &= \mathbf{E}_t \\ \mathbf{A}^T\mathbf{A}\mathbf{v} &= \mathbf{A}^T\mathbf{E}_t \\ \mathbf{v} &= \left[\mathbf{A}^T\mathbf{A}\right]^{-1}\mathbf{A}^T\mathbf{E}_t \end{aligned}$$

where $\left[\mathbf{A}^T\mathbf{A}\right]^{-1}\mathbf{A}^T$ is the pseudo inverse. Instead of solving the linear equations above, the optical flow can be estimated using least-squares fit.

$$\min \sum_i \left(E_{xi}u + E_{yi}v + E_{ti}\right)^2 \tag{4.3}$$

In order to minimize the square error, (4.3) is differentiated and set equal to zero. The partial derivative with respect to u yields

$$\begin{aligned}\frac{\partial}{\partial u}\sum_i \left(E_{xi}u + E_{yi}v + E_{ti}\right)^2 &= 0\\ \sum_i \left(E_{xi}u + E_{yi}v + E_{ti}\right)E_{xi} &= 0\\ \sum_i E_{xi}^2 u + \sum_i E_{yi}E_{xi}v &= -\sum_i E_{ti}E_{xi}\end{aligned}$$

Similarly, the partial derivative with respect to v

$$\sum_i E_{xi}E_{yi}u + \sum_i E_{yi}^2 v = -\sum_i E_{ti}E_{yi}$$

The two partial derivatives can be written as a linear system on matrix form

$$\begin{bmatrix} \sum_i E_{xi}^2 & \sum_i E_{yi}E_{xi} \\ \sum_i E_{yi}E_{xi} & \sum_i E_{yi}^2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum_i E_{ti}E_{xi} \\ -\sum_i E_{ti}E_{yi} \end{bmatrix}$$

Here it is easy to see that the matrix in front of the velocity vector is a square matrix, and consequently can be inverted.

$$\begin{aligned}\begin{bmatrix} u \\ v \end{bmatrix} &= \begin{bmatrix} \sum_i E_{xi}^2 & \sum_i E_{yi}E_{xi} \\ \sum_i E_{yi}E_{xi} & \sum_i E_{yi}^2 \end{bmatrix}^{-1}\begin{bmatrix} -\sum_i E_{ti}E_{xi} \\ -\sum_i E_{ti}E_{yi} \end{bmatrix}\\ &= \frac{1}{\sum_i E_{xi}^2 \sum_i E_{yi}^2 - \left(\sum_i E_{yi}E_{xi}\right)^2}\begin{bmatrix} \sum_i E_{yi}^2 & -\sum_i E_{yi}E_{xi} \\ -\sum_i E_{yi}E_{xi} & \sum_i E_{xi}^2 \end{bmatrix}\begin{bmatrix} -\sum_i E_{ti}E_{xi} \\ -\sum_i E_{ti}E_{yi} \end{bmatrix}\end{aligned}$$

Hence, the optical flow $\mathbf{v} = \begin{bmatrix} u & v \end{bmatrix}^T$ can be found as

$$\begin{aligned}u &= \frac{-\sum_i E_{yi}^2 \sum_i E_{ti}E_{xi} + \sum_i E_{yi}E_{xi} \sum_i E_{ti}E_{yi}}{\sum_i E_{xi}^2 \sum_i E_{yi}^2 - \left(\sum_i E_{yi}E_{xi}\right)^2}\\ v &= \frac{\sum_i E_{yi}E_{xi} \sum_i E_{ti}E_{xi} - \sum_i E_{xi}^2 \sum_i E_{ti}E_{yi}}{\sum_i E_{xi}^2 \sum_i E_{yi}^2 - \left(\sum_i E_{yi}E_{xi}\right)^2}\end{aligned}$$

Weighted Lucas-Kanade method

In the Lucas-Kanade algorithm presented above, each of the pixels have equal weight. It ids reasonable to assume that the pixels closer to the center of the window computes the optical flow more accurately. Hence, the least-squares fit can be modified to include weight on each of the pixels within the window

$$\min \sum_i w_i \left(E_{xi}u + E_{yi}v + E_{ti}\right)^2$$

where w_i is a weight for pixel p_i. Using the same procedure as for the least-squares, the optical flow can be calculated as

$$u = \frac{-\sum_i w_i E_{yi}^2 \sum_i w_i E_{ti} E_{xi} + \sum_i w_i E_{yi} E_{xi} \sum_i w_i E_{ti} E_{yi}}{\sum_i w_i E_{xi}^2 \sum_i w_i E_{yi}^2 - \left(\sum_i w_i E_{yi} E_{xi}\right)^2}$$
$$v = \frac{\sum_i w_i E_{yi} E_{xi} \sum_i w_i E_{ti} E_{xi} - \sum_i w_i E_{xi}^2 \sum_i w_i E_{ti} E_{yi}}{\sum_i w_i E_{xi}^2 \sum_i w_i E_{yi}^2 - \left(\sum_i w_i E_{yi} E_{xi}\right)^2}$$

Lucas-Kanade method with pyramids

Both variations of the Lucas-Kanade algorithm derived above have one problem in common. In cases where the motion is large, the algorithms fail to produce accurate optical flow. Pyramids are a very useful representation of an image and is created by using multiple copies of the same image. Each level in the pyramid is 1/4 of the size of the previous level. The lowest level of the pyramid is the image with the highest resolution, often the raw image. Consequently, the highest level is the image with the lowest resolution. For a visual representation of the pyramid, see Figure 4.3.

To compute optical flow with Lucas-Kanade pyramids, Algorithm 4.1 can be used [64]

Algorithm 4.1 Lucas-Kanade method with pyramids

$n \leftarrow$ number of levels in pyramid
for $i \leftarrow n, 1$ **do**
  Take flow u_{i-1}, v_{i-1} from level $i-1$
  Bilinear interpolate it to create u_i^*, v_i^* matrices of twice resolution for level i
  Multiply u_i^*, v_i^* by 2
  Compute E_t from a block displaced by $u_i^*(x,y)$, $v_i^*(x,y)$
  Apply LK to get $u_i'(x,y)$, $v_i'(x,y)$ (the correction in flow)
  Add corrections u_i', v_i', i.e. $u_i = u_i^* + u_i'$, $v_i = v_i^* + v_i'$
end for

The algorithm is explained graphically in Figure 4.4

4.1.3 Local navigation

The optical flow algorithms derived above have many applications in local navigation. In this section three different applications of the optical flow are discussed. Firstly, the velocity vector received by an optical flow algorithm and how it can be used in object tracking is presented. Secondly, the case of object avoidance is discussed. Thirdly, show the optical flow of two features can be used to estimate the angular velocity of an object is shown.

Object tracking

The velocity vector received by an optical flow algorithm is [29]

$$F = -\boldsymbol{\omega} + \frac{\mathbf{v}^c}{d} \sin(\theta) \tag{4.4}$$

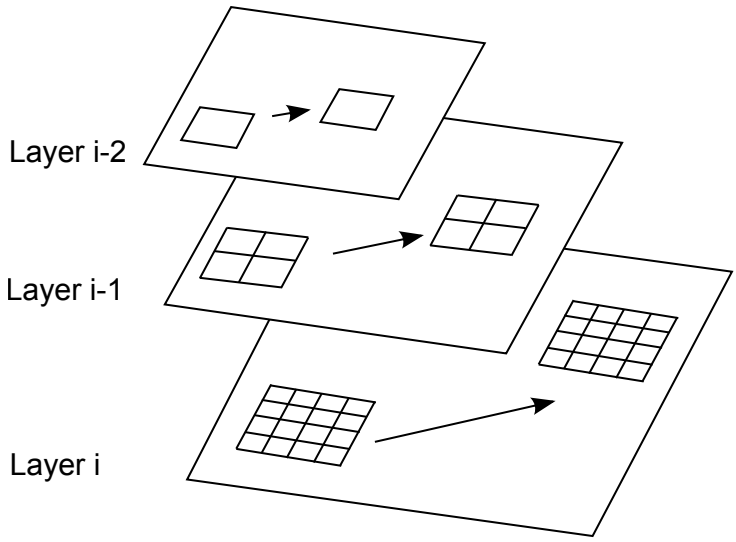


Figure 4.3: Sketch of a pyramid with three levels. The number of pixels is reduced with a factor of four between each level. Based on [43]

where F is the optical flow produced by two kinds of motion; rotational, $F_{rot} = -\boldsymbol{\omega}$, and translational, $F_{tra} = (\mathbf{v}^c/d)\sin(\theta)$. Furthermore, $\boldsymbol{\omega}$ is the angular velocity of the observer and θ is the angle between observer and object (review Figure 4.1). The last remaining terms, $\mathbf{v}^c$ and d, are the relative metric velocity and distance to the object, respectively.

The relative velocity vector gives an indication of how fast and in which direction the object is moving relative to the UAV. This can be used to design a control law capable of tracking the target, i.e. make the optical flow approach zero.

Obstacle avoidance

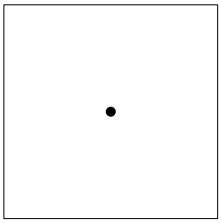
The optical flow can also be used to avoid obstacles. With the help of optical flow, an observer will be notified when an object comes closer or further away. This information can be used to avoid objects by seeking out where the optical flow is the smallest. This is particularly useful indoors, as the space is limited by walls and ceiling. In Figure 4.5a, an UAV uses optical flow to navigate through a narrow corridor. Figure 4.5b shows an application of the optical flow obstacle avoidance in an outdoor environment.

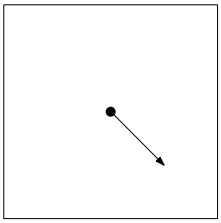
Estimate angular velocity

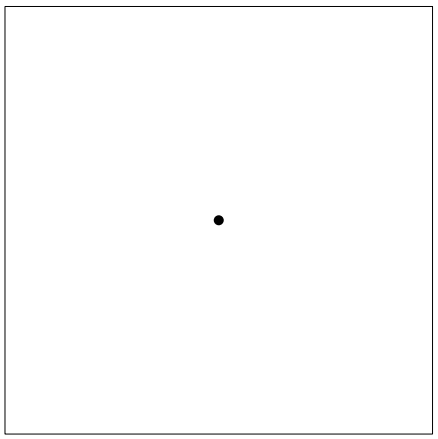
If the object taken into consideration is rotating, it is possible to estimate the angular velocity and feedforward the estimate to a controller. This will enable the system to predict where the object will be at the next frame and do the necessary corrections, thus achieve a more robust tracking controller. The angular velocity can be found as

$$\boldsymbol{\omega} = \frac{\mathbf{v}_\perp}{r} \tag{4.5}$$

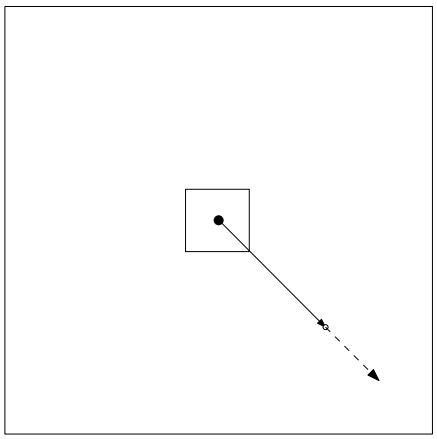
where r is the distance from the center of rotation to the feature the UAV is tracking, and $\mathbf{v}_\perp$ is the perpendicular velocity. Since a camera is able to capture pictures at a high frame rate, the displacement of the object between frames will be small. Consequently,



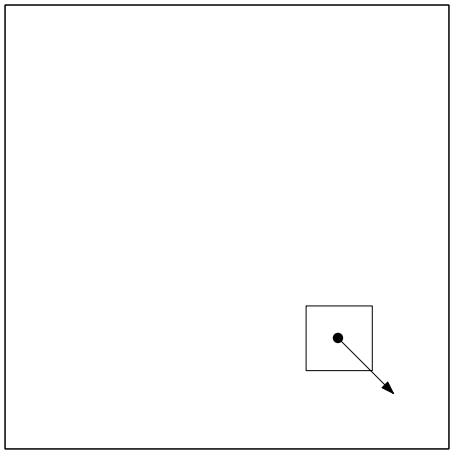




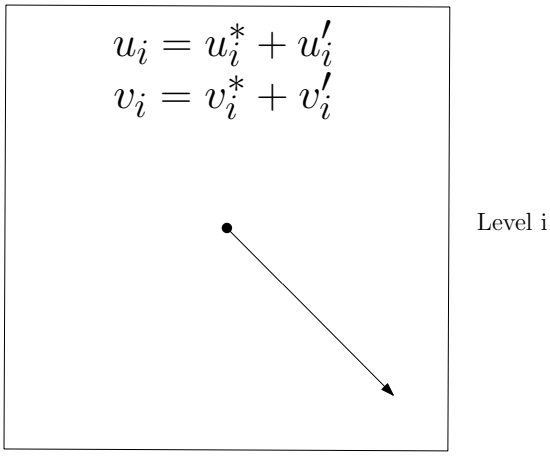
(a) The feature in level $i-1$ is propagated to level i. Level i has four times the resolution as level $i-1$. The optical flow in level $i-1$ is calculated using Lucas-Kanade



(b) The optical flow obtained in level $i-1$ is propagated to level i by multiplying with two. The resulting optical flow is denoted u_i^*, v_i^*. The dotted line represent the true optical flow.

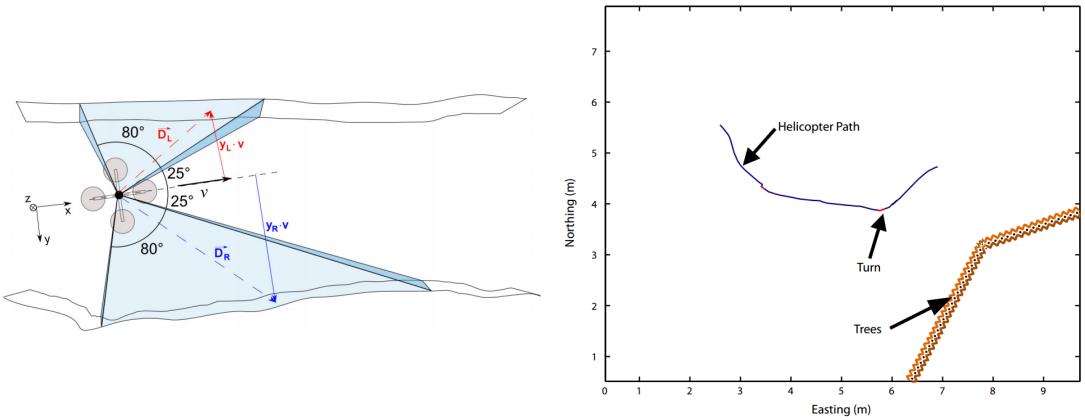


(c) The residual optical flow, u_i', v_i', is calculated by applying Lucas-Kanade in level i with a window centred around u_i^*, v_i^*



(d) The optical flow is achieved by adding the optical flow from level $i-1$ propagated to level i and the residual optical flow.

Figure 4.4: Step by step Lucas-Kanade with pyramids, idea [64]



(a) A quadcopter navigating in a corridor with two cameras. The UAV is controlled such that the optical flow of each camera is equal. Courtesy of [80]

(b) A helicopter avoids trees by turning when the calculated optical flow is below a threshold. Courtesy of [29]

Figure 4.5: Two different applications of obstacle avoidance

we can assume that $\mathbf{v} \approx \mathbf{v}_\perp$, see Figure 4.6. However, $\mathbf{v}$ is not only the velocity vector caused by the wind turbine, but also the UAV, see Figure 4.9. Hence we can write the relative velocity vector as

$$\mathbf{v} = \mathbf{v}_O + \mathbf{v}_{UAV} \tag{4.6}$$

where $\mathbf{v}_O$ and $\mathbf{v}_{UAV}$ is the velocity vector caused by the object and the UAV, respectively. In order to use the velocity vector to estimate the angular velocity, the contribution from the UAV needs to be cancelled out. Let $\mathbf{f}$ be the acceleration of the UAV and t_s be the time between two frames. Thus, the estimated velocity of the UAV can be written

$$\hat{\mathbf{v}}_{UAV} = \mathbf{f} \cdot t_s$$

Introducing the above expression in (4.6) yields

$$\mathbf{v}_O = \mathbf{v} - \mathbf{f} \cdot t_s$$

In order to estimate the angular velocity, the radius and thus also the origin is needed. The origin can be found by tracking two feature points $[(x_1, y_1), (x_2, y_2)]$ and adding the optical flow vector to create two additional points $[(x_3, y_3), (x_4, y_4)]$. By drawing a line between the two features in each point, the origin can be found where the lines intersect, see Figure 4.7. Mathematically, it is calculated as [74]

$$O_1 = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & x_1 - x_2 \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & x_3 - x_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}} \qquad O_2 = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & y_1 - y_2 \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & y_3 - y_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}} \tag{4.7}$$

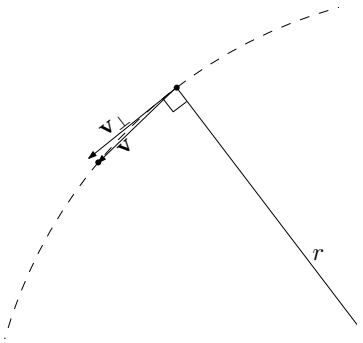


Figure 4.6: Comparison of the perpendicular velocity vector and the metric velocity obtained by the EKF. The dotted line is the trajectory of a rotating object

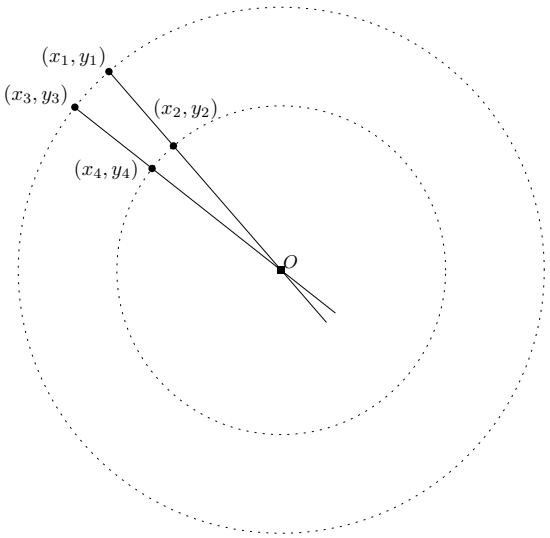


Figure 4.7: The origin is where the lines, created by two features with added optical flow, intersects each other. The dashed lines are the trajectories of the two features on a rotating object

where O_1 and O_2 is the x and y-value of the origin, respectively. Using simple Pythagoras, the radii are found

$$r_1 = \sqrt{(x_1 - O_1)^2 + (y_1 - O_2)^2} \qquad r_2 = \sqrt{(x_2 - O_1)^2 + (y_2 - O_2)^2} \tag{4.8}$$

The angular velocity is then calculated for each of the features using (4.5) and then taking the average, i.e. $\omega = (\omega_1 + \omega_2)/2$. The procedure is summarized in algorithm 4.2 and graphically in Figure 4.8.

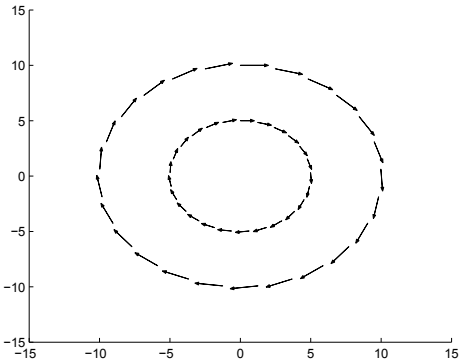
Algorithm 4.2 Estimate angular velocity

for each frame **do**
- Choose two features to track
- Store the position of the features and the added optical flow positions
- Estimate the origin using (4.7)
- Estimate the radii using (4.8)
- Estimate the angular velocity using (4.5)

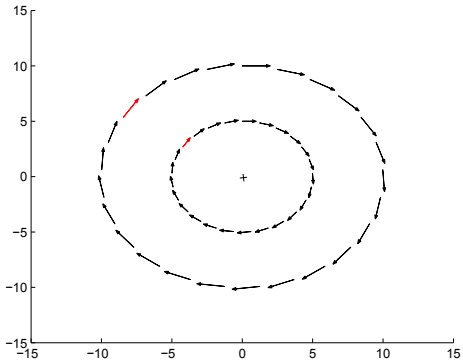
end for

4.2 Hough transform

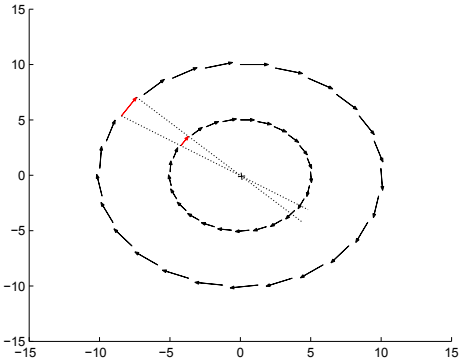
In cases where there is a lack of features to track and the object consists of one or several straight lines, Hough transform can be used to calculate the desired velocity vector. This section will first give a short overview of Hough transform and how it can be used to detect straight lines in an image. It will then include a discussion of how Hough transform can be used in local navigation.



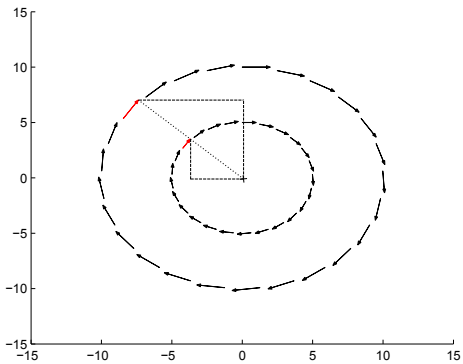
(a) The velocity vectors received from an optical flow algorithm. Two features were tracked for a full revolution



(b) The two feature points were extracted. The optical flow vector for these two features are highlighted in red



(c) Two lines were drawn from the four positions and the origin were found at the intersection, using (4.7)



(d) The radii were found by using simple Pythagoras and the angular velocity was estimated using (4.5)

Figure 4.8: Step by step estimating angular velocity from optical flow measurement

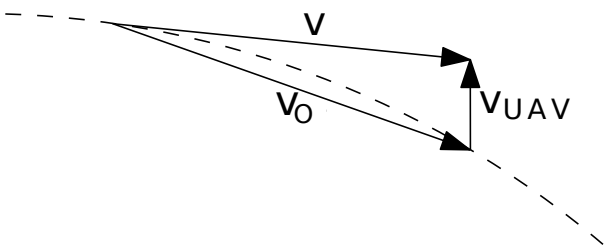


Figure 4.9: The estimated velocity vector is a combination of wind turbine and UAV velocity. The dashed line is the trajectory of a rotating object

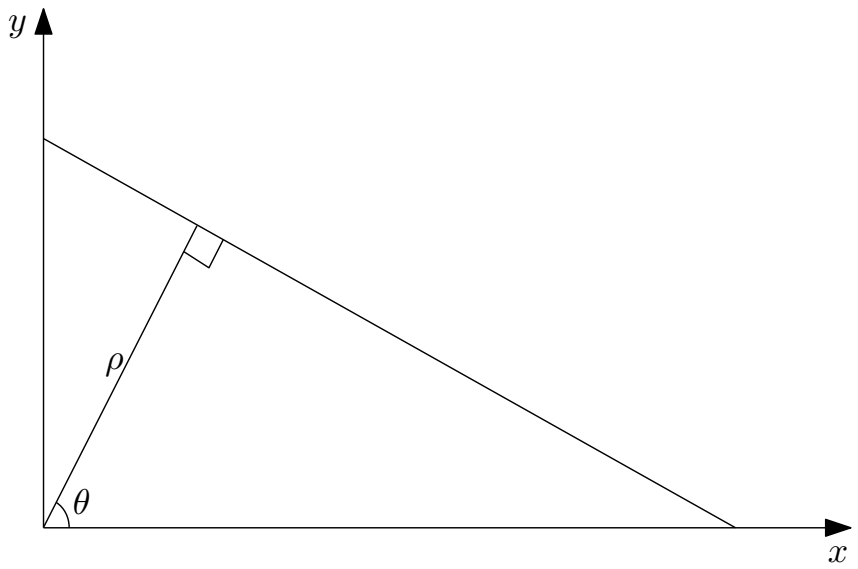


Figure 4.10: A line can be expressed in both Cartesian (x, y) and polar (ρ, θ) coordinates

4.2.1 Overview

In 1962, Paul Hough released a patent on recognizing complex patterns using a slope-intersect parameter space [28]. Duda and Hart investigated this further and presented a paper where they use polar coordinates to detect any parametric curves, e.g. line, circle, ellipse [13]. While edge detection will detect the line, it will not give a mathematical expression. The mathematical expression is necessary to extract information about the line, e.g. angle, length and intersection between lines.

4.2.2 Detecting straight lines

Given a set of edge points, the Hough transform finds the line(s) which best fit the data. A straight line can be described mathematically as

$$y = mx + c \tag{4.9}$$

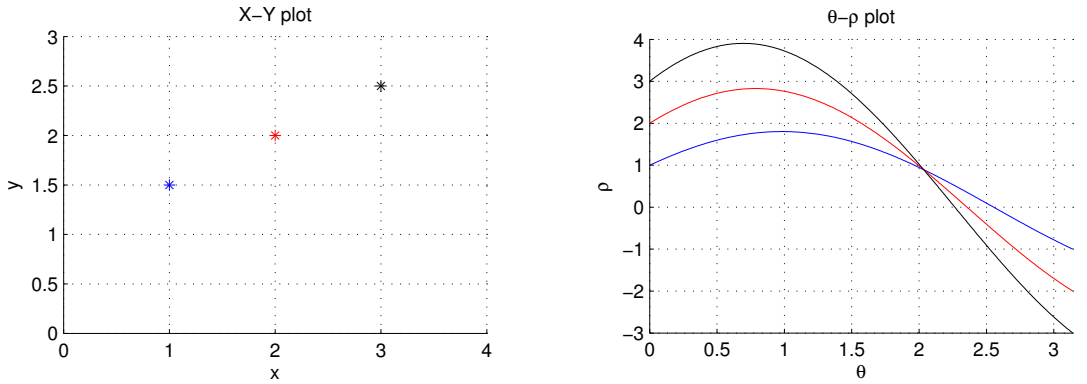
where m is the slope and c is the intersect value. Not every line can be described by this equation as the slope tends to be infinite. Instead, (4.9) is written in polar coordinates

$$\rho = xcos\theta + ysin\theta \tag{4.10}$$

where ρ is the distance from the image corner perpendicular to the line and $\theta \in (-\pi/2, \pi/2)$ is the angle, see Figure 4.10. For each pixel detected by the edge detection algorithm, (4.10) will create a sinusoid. The sinusoids will intersect for those pixels that belongs to the same straight line, i.e. have the same ρ and θ value, see Figure 4.11. These ρ and θ values will get a vote, and the pairs with most votes belongs to a straight line. The number of maxima is the number of straight lines. The procedure is summarized in Algorithm 4.3.

4.2.3 Local navigation

Hough transform can be used in cases where the object consists of multiple straight lines, e.g. wind turbines and building structures. By using Hough transform on a wind turbine



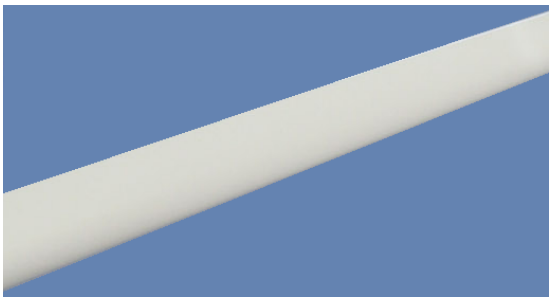
(a) Three points extracted from a edge detection algorithm

(b) Hough transform of the three points

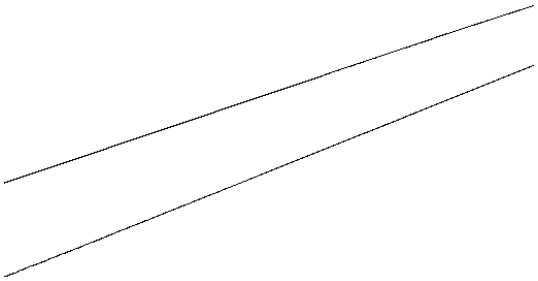
Figure 4.11: Given a set of points in Cartesian coordinates (a), the Hough transform can find the equation in polar coordinates, (b). The sinusoid intersects in $\rho = 0.8944$ and $\theta = 2.034$ corresponding to the line $y = 0.5x + 1$

Algorithm 4.3 Hough transform for fitting straight lines

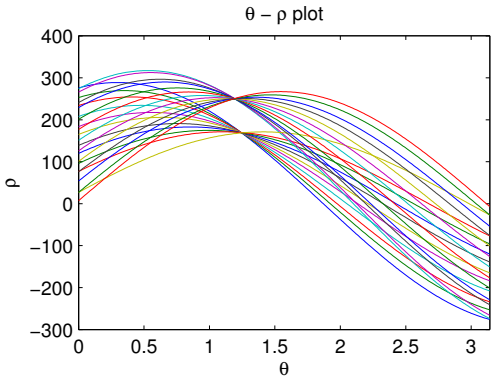
Quantize the two-dimensional parameter space $P[\rho_{min}, \ldots, \rho_{max}, \theta_{min}, \ldots, \theta_{max}]$
n number of edge pixels
for i=0, i<n, i++ **do**
  for $(\theta = \theta_{min}, \theta \leq \theta_{max}, \theta++)$ **do**
    $\rho = x_i \cos(\theta) + y_i \sin(\theta)$
    $P[\rho, \theta] += 1$
  end for
end for
Find the local maxima on the parameter space



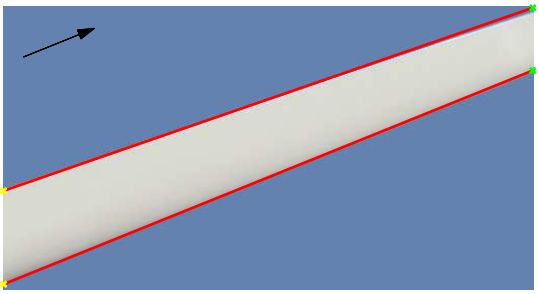
(a) Wind turbine blade



(b) Edge detection



(c) Hough transform in a (ρ, θ) plot

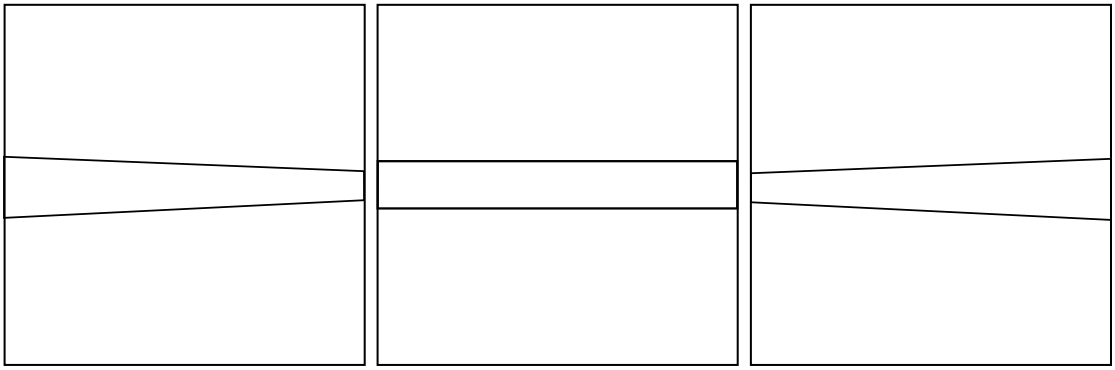


(d) Plot of the detected lines in original image

Figure 4.12: Wind turbine blade before (a) and after (b) edge detection using the canny edge detection in MATLAB with threshold [0.02 0.09]. The Hough transform is shown in (c) and merged on the original image in (d). The arrow in (d) points in the direction of the blade and will be the desired velocity vector for the UAV

blade, the blade can be described mathematically and thus create a vector in the direction of the blade. The vector can be used in a pure pursuit guidance (see section 6.1), where the length of the vector, i.e. speed of the UAV, can be set by an operator. Figure 4.12 shows the Hough transform performed on a wind turbine blade.

In addition, Hough transform can be used to detect the angle between straight lines, which again can be used in yaw control. As illustrated in Figure 4.13, the angle between the edges of the blade will change according to the orientation of the camera. By using a priori information about the object, the Hough transform can detect errors in yaw and send this information back to a yaw controller. It thus ensures that the UAV is perpendicular to the object at all times. This is applicable in both the case of wind turbine inspection and building inspection, as a building structure consist of several straight line with known angle, e.g. windows and bricks.



(a) Positive heading (b) Neutral heading, camera perpendicular to the blade (c) Negative heading

Figure 4.13: The angle between the edges of the blade changes according to the camera rotation.

Chapter 5

Filter design

Chapter 3 developed the necessary equations for describing the equation of motion between the UAV and an object. In Chapter 4 the output from a optical flow algorithm was discussed. In this chapter the information from these two chapters are combined to create an observer capable of estimating the metric, non-scaled velocity vector and the distance.

5.1 Introduction

In the last five decades, many research areas, e.g. robotics, weather forecasting and navigation, have investigated how to estimate the system states from noisy measurement inputs. In 1960, Rudolph E. Kalman published his famous paper on filtering linear problems in discrete time [32]. The Kalman Filter (KF) is a recursive filter that estimates unmeasured states and the process output of a linear system. The filter is optimal with respect to minimizing the mean square error. However, most systems in real life are nonlinear and in many applications it is not sufficient to linearize the model. Therefore, additional approaches to estimate the unknown states in a nonlinear system have been developed.

5.2 Extended Kalman filter

The extended Kalman filter (EKF) is a nonlinear version of the KF and is widely used today in navigation and object tracking. The EKF was developed by NASA Ames Research Center in the early sixties during studies of navigation and control for the Apollo space capsule [45]. The EKF linearizes the nonlinear system about its best estimate at each time step. However, linearization leads to loss of the optimality property. This means that the EKF can diverge if the system equations are inaccurate or the initial state predictions are too far from the true states.

Consider a general nonlinear system on the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathbf{w} \tag{5.1}$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) + \mathbf{v} \tag{5.2}$$

where $\mathbf{f}(\mathbf{x}, \mathbf{u})$ is a nonlinear vector field and $\mathbf{h}(\mathbf{x})$ is a nonlinear measurement function. The process and measurement noise, $\mathbf{w}$ and $\mathbf{v}$, are assumed to be zero mean Gaussian white noise with covariance matrix $\mathbf{Q}_k = \mathbf{Q}_k^T > 0$ and $\mathbf{R}_k = \mathbf{R}_k^T > 0$, respectively. The EKF algorithm for estimating the states is presented below and graphically in Figure 5.1

1. The algorithm first computes the Kalman gain at time k, $\mathbf{K}_k$. This gain will minimize the mean-square estimation error. The Kalman gain can be calculated as

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T \left[\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k \right]^{-1}$$

   where $\mathbf{P}_k^-$ is the predicted covariance estimate and $\mathbf{H_k}$ is the linearized measurement vector given as

$$\mathbf{H}_k = \left. \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}_k} \right|_{\mathbf{x}_k = \hat{\mathbf{x}}_k^-} \tag{5.3}$$

2. The gain is then used to update the estimates with the new measurements.

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_k^-) \right)$$

   Here $\hat{\mathbf{x}}_k^-$ is the predicted state estimate, $\mathbf{z}_k$ is the new measurement and $\mathbf{h}(\hat{\mathbf{x}}_k^-)$ is the nonlinear measurement function.

3. The Kalman gain is also used to compute the error covariance for the updated estimate.

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k \right) \mathbf{P}_k^- \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k \right)^T + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T$$

4. The estimated states and covariance matrix is then projected ahead to time $k+1$

$$\hat{\mathbf{x}}_{k+1}^- = \mathcal{F}(\hat{\mathbf{x}}(k))$$

$$\mathbf{P}_{k+1}^- = \mathbf{\Phi}_k \mathbf{P}_k \mathbf{\Phi}_k^T + \mathbf{Q}_k$$

   where $\mathcal{F}(\hat{\mathbf{x}}(k))$ and $\mathbf{\Phi}_k$ are obtained using forward Euler integration

$$\begin{aligned} \mathcal{F}(\hat{\mathbf{x}}(k)) &= \hat{\mathbf{x}}_k + h \left[\mathbf{f}(\hat{\mathbf{x}}_k, \mathbf{u}_k) \right] \\ \mathbf{\Phi}_k &= \mathbf{I} + h \left. \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}_k} \right|_{\mathbf{x}_k = \hat{\mathbf{x}}_k} \end{aligned} \tag{5.4}$$

These four steps are then repeated ad infinitum, or as long as there are measurements available. If there are no measurements available, the EKF is able to predict the state estimate purely on the system model. When a new measurement is available again, it will correct the estimates accordingly. For a full derivation of the EKF, the reader is referred to [6].

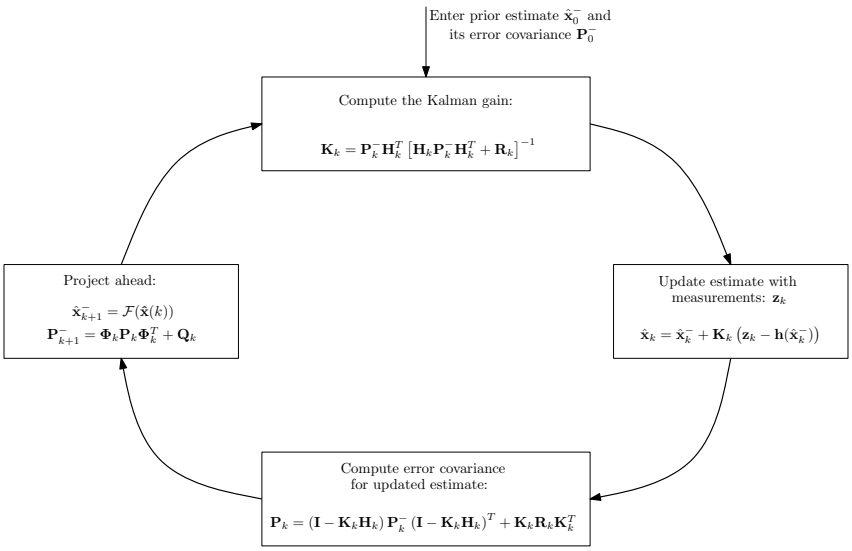


Figure 5.1: The extended Kalman filter loop. Based on Figure 4.1 p.147 in [6]

5.3 Implementation

It was shown in section 4.1 that the output from an optical flow algorithm can be written as

$$F = -\omega + \frac{\mathbf{v}^c}{d} \sin(\theta) \tag{5.5}$$

where F is composed of translational, F_{tra}, and rotational, F_{rot}, optical flow, ω is the angular velocity and θ is the angle between the UAV and the object. The angular velocity is measured by the IMU and can be used to cancel out the rotational contribution of the optical flow. Hence, only the translational flow

$$F_{tra} = \frac{\mathbf{v}^c}{d} \sin(\theta)$$

is measured. As shown in section 4.2.3 the UAV can be controlled to always face directly towards the blade. However, this will only work in the case where the blade is horizontal. When the blade is vertical, rotation about yaw will not have an impact on the angle. Thus, only the information provided by the autopilot is available. These two methods can be combined by using the camera to aid the heading estimate provided by the IMU. As a result, a simple heading controller can be implemented in order to make the camera and blade perpendicular to each other, i.e. $\sin(\theta) \approx 1$. With these assumptions, the optical flow output (5.5) can be simplified to

$$F = \frac{\mathbf{v}^c}{d}$$

Defining the state vector $\mathbf{x} = [\mathbf{v}^c \;\; d]^T$ and with the system equations (3.5) and (3.6), the system can be written in the form (5.1)-(5.2) with

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{R}_b^c \left(\mathbf{f}^b + \mathbf{R}_n^b \mathbf{g}^n + \mathbf{S}^2(\boldsymbol{\omega}^b)\mathbf{p}_{bc}^b\right) - \mathbf{S}(\boldsymbol{\omega}^c)\mathbf{v}^c \\ \mathbf{n}^c \mathbf{v}^c \end{bmatrix}, \qquad \mathbf{h}(\mathbf{x}) = \frac{\mathbf{v}^c}{d}$$

The Jacobians are calculated according to (5.3)-(5.4)

$$\begin{aligned}\mathbf{\Phi}_k = \mathbf{I}_4 + h\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}_k} &= \mathbf{I}_4 + h\begin{bmatrix} \dfrac{\partial}{\partial \mathbf{v}^c}(-\mathbf{S}(\boldsymbol{\omega}^c)\mathbf{v}^c) & \dfrac{\partial}{\partial d}(-\mathbf{S}(\boldsymbol{\omega}^c)\mathbf{v}^c) \\ \dfrac{\partial}{\partial \mathbf{v}^c}(\mathbf{n}^c\mathbf{v}^c) & \dfrac{\partial}{\partial d}(\mathbf{n}^c\mathbf{v}^c) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I}_3 - h\mathbf{S}(\boldsymbol{\omega}^c) & \mathbf{0}_{3\times 1} \\ h\mathbf{n}^c & 1 \end{bmatrix}\end{aligned}$$

$$\begin{aligned}\mathbf{H}_k = \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}_k} &= \begin{bmatrix} \dfrac{\partial}{\partial \mathbf{v}^c}\left(\dfrac{\mathbf{v}^c}{d}\right) & \dfrac{\partial}{\partial d}\left(\dfrac{\mathbf{v}^c}{d}\right) \end{bmatrix} \\ &= \begin{bmatrix} \dfrac{1}{d}\mathbf{I}_3 & -\dfrac{\mathbf{v}^c}{d^2} \end{bmatrix}\end{aligned}$$

Chapter 6

Guidance

The following chapter describes the guidance systems for inspecting a wind turbine or building for damages. The guidance is based on the two-point guidance scheme pure pursuit with and without feed forward of angular velocity. The last section describes the different PID controllers used to calculate the desired Euler angles, as well as throttle. This thesis assumes that there are no acrobatic maneuvers executed. For this reason, the coupling between the four controllers is ignored.

6.1 Pure pursuit guidance

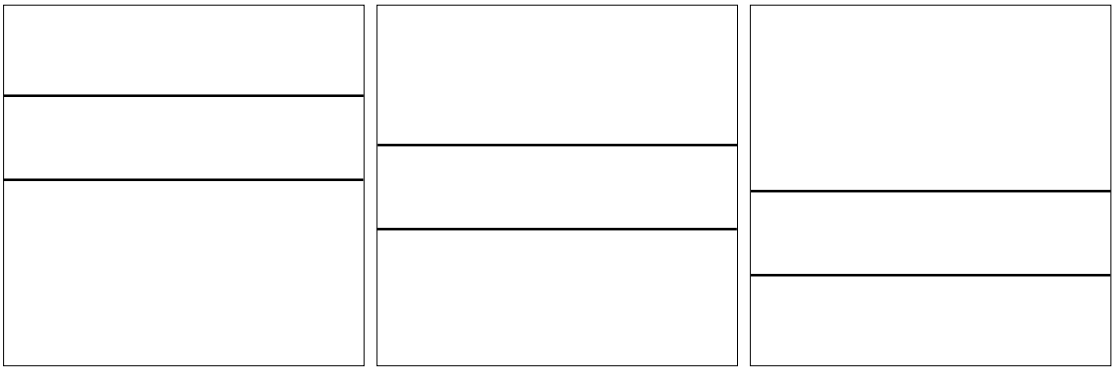
In pure pursuit guidance the UAV creates a line-of-sight (LOS) vector between the UAV itself and the object it is tracking, see Figure 6.2. The desired velocity, $\mathbf{v}_d$, is aligned with this vector and can be calculated as [18]

$$\mathbf{v}_d = -\kappa \frac{\tilde{\mathbf{p}}}{\| \tilde{\mathbf{p}} \|} \tag{6.1}$$

where $\tilde{\mathbf{p}} = \mathbf{p} - \mathbf{p}_t$ is the distance between the UAV and object, and $\| \cdot \|$ is the Euclidean length of the vector. The design parameter $\kappa > 0$ is chosen arbitrary. For a more detailed description of the pure pursuit guidance, the reader is referred to [18] and [78].

The vector received from one of the previously mentioned computer vision algorithms can be interpreted as a pure pursuit vector. The vector starts at the camera and with a direction determined by either optical flow or Hough transform. The vector is continuously updated as long as there is new information available from the camera. Since the position of the UAV and the object is unknown, the desired speed in (6.1) is not possible to calculate. Instead, the velocity of the UAV can be controlled by scaling the length of the vector, i.e. a longer vector yields higher desired speed.

In the wind turbine scenario, the UAV can also be controlled to always have the blade in the center of the image. This is achieved by looking at the area above and below the blade, see Figure 6.1. A larger area below implies that the UAV is too low and needs to move upwards. This information can be fed into a controller which will make the necessary adjustments, e.g. compensate the LOS vector by either adding or subtracting the necessary corrections in z-direction.



(a) UAV too low (b) Perfect (c) UAV too high

Figure 6.1: The area above and below the blade

6.2 Pure pursuit guidance with feed forward

In section 4.1.3, it was shown how the angular velocity of a rotating object can be estimated by tracking two or more features. To take advantage of the estimated angular velocity, the pure pursuit guidance presented above needs to be modified. Instead of trying to track where the object is at the moment, it will track where the object will be in the future. Hence, the updated pure pursuit guidance can be written as

$$\mathbf{v}_d^* = -\kappa \left(\frac{\tilde{\mathbf{p}}}{\| \tilde{\mathbf{p}} \|} + \boldsymbol{\gamma} \right)$$

where $\boldsymbol{\gamma}$ is the displacement vector added as a result of including feedforward of angular velocity, see Figure 6.3.

6.3 PID controller

In order to follow the desired velocity vector, a controller is needed. A PID controller is one of the most popular controllers due to its simplicity, adaptability and performance in cases where the system parameters are unknown. The goal of the PID controller is to minimize the error defined as

$$e(t) = r(t) - y(t)$$

where $r(t)$ is the set point and $y(t)$ is the measured state. In order to achieve this, the controller consist of three terms; proportional, integral and derivative. The proportional term multiplies the error with a proportional coefficient and is dependent on the current error. However, the proportional controller will lead to an offset or droop. The integral term adds up the previous errors and thus removes the offset. An integral controller is included when the process is exposed to steady disturbances, e.g. ocean currents or gravity. In most cases, a PI-controller will produce satisfactory results. However, in processes where there are little measurement noise and a measurement of the derivative is available, a derivative term is added. The added term decreases the settling time and improves stability. By using the derivative of the error, the controller is able to predict and correct future errors. The affect of including the different terms are shown in Figure 6.4.

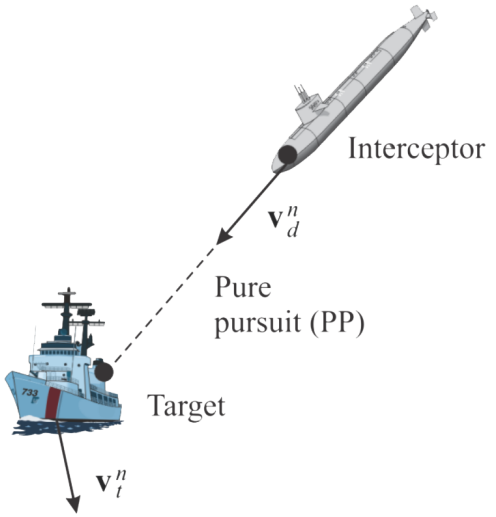


Figure 6.2: Desired velocity vectors for pure pursuit guidance. Courtesy of [18]

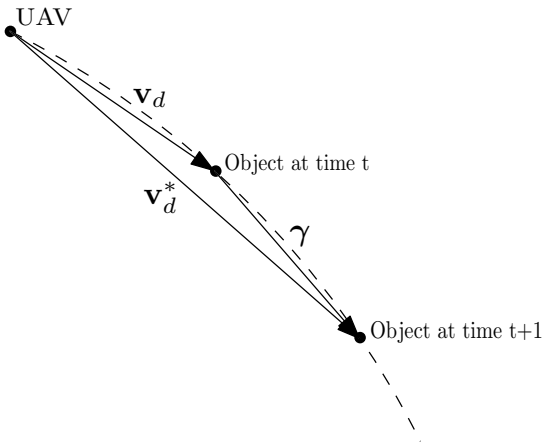
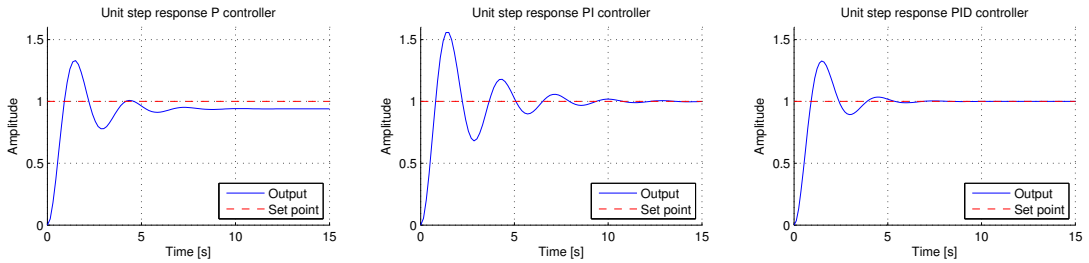


Figure 6.3: The UAV align the desired velocity vector, $\mathbf{v}_d$, with the LOS vector. The object's position at time $t+1$ is projected using the displacement vector, γ, calculated using the estimated angular velocity. The dotted line is the trajectory of the wind turbine



(a) P controller (b) PI controller (c) PID controller

Figure 6.4: Illustration of the affect of adding a P, I and D term in the controller

The PID controller will be the sum of the proportional, integral and derivative controller and can be described as

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \dot{e} \tag{6.2}$$

where K_p, K_i and K_d are user-defined gains. The PID controller is described graphically in Figure 6.5.

6.3.1 Pitch

As previously mentioned, the distance to the wind turbine blade or a building is measured using a sonar. The goal is to control the UAV so that it always has a desired distance to the object. This can be achieved with the use of a PD controller with the error defined as

$$e = d_{ref} - d_{sonar}$$

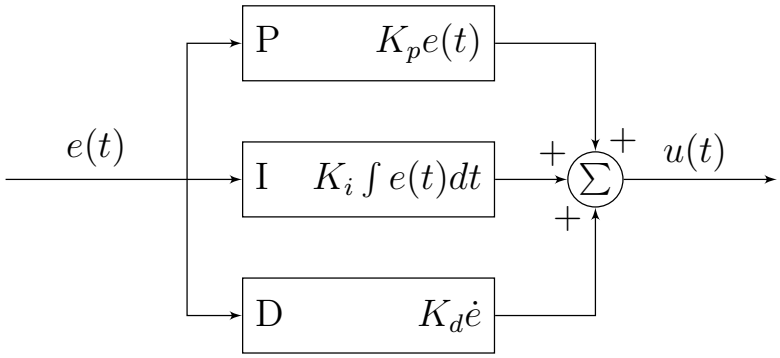


Figure 6.5: Block diagram of a PID controller

where d_{ref} is the desired distance to the object and d_{sonar} is the distance measured by the sonar. By using the PD controller in (6.2) ($K_i = 0$) the error, e, will go to zero. The derivative of the distance is achieved through numerical differentiation of the sonar value. Since the desired distance is stationary, the derivative of the error can be written

$$\dot{e} = \dot{d}_{ref} - \dot{d}_{sonar} = -\dot{d}_{sonar} = -\frac{d_{sonar} - d_{sonar\_prev}}{dt}$$

where d_{sonar_prev} is the previous sonar measurement. The output of the controller is the desired pitch value. To ensure that the UAV does not receive desired pitch values that could flip it, the controller saturates the desired pitch values.

6.3.2 Roll

The roll controller receives the desired velocity vector and decomposes the vector in y (east) and z (down) direction. The y-direction is controlled by roll and the desired roll angle can be calculated with a PD regulator. The controller receives the estimated velocity from the optical flow algorithm and the derivative from the accelerometer. The output of the controller is the desired roll angle in either radians or degrees, dependent on the autopilot. Similarly to the pitch controller, the roll controller also features saturation on the desired angle.

6.3.3 Yaw

The angular velocity is used in the in-built yaw controller, so a P controller will be sufficient. An I-term can be added to reduce static deviation. As shown in section 4.2.3, the Hough transform can detect the angles between the blade or straight lines on a building. This information can be fed to the controller, which will calculate the necessary corrections. The output of the controller will be the desired yaw angle given in radians.

6.3.4 Height

The height controller uses the z-component of the decomposed vector to compute the desired relative height, i.e. how much the UAV should move in relation to the current height. Unlike the controllers above, the height controller uses a PID controller. The integral term is included due to the constant force of gravity forcing the UAV down, and

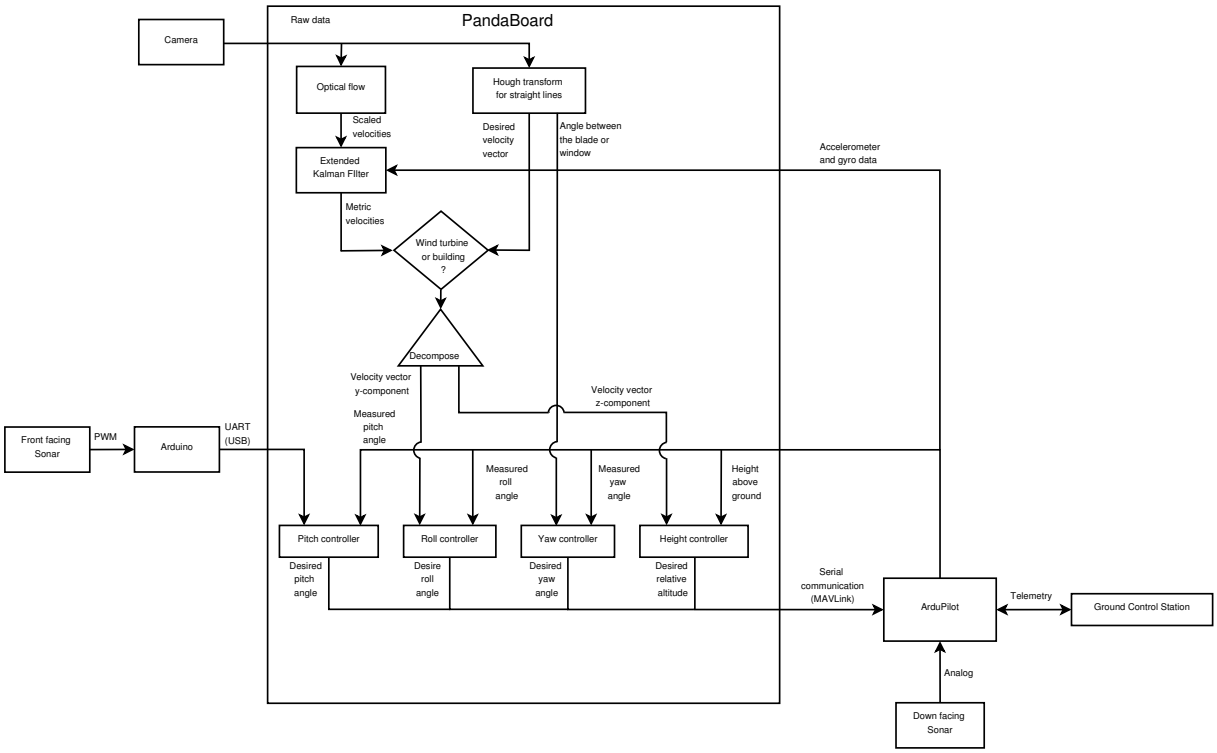


Figure 6.6: Block diagram of the different components and their interactions

to provide enough thrust when the battery voltage and weight of the payload changes. The lack of an I-term will lead to static deviation, as shown above. The output of the controller is the desired change of the current altitude. The altitude is given by the barometer which will be initialised at take-off. The barometer measures the barometric pressure and calculates the relative height as follows [75]

$$h_{rel} = \frac{T_0}{\Gamma}\left[\left(\frac{P_k}{P_0}\right)^{-\Gamma\frac{R}{g}} - 1\right]$$

where T_0 and P_0 are the initial temperature and pressure, respectively. The initial pressure and temperature are measured at start up by the barometer and the thermometer. Furthermore, the Γ, R and g is the lapse rate, universal gas constant and gravity constant, respectively. Another possibility is to use a sonar to measure the relative height. In section 9.3.3 the barometer and sonar is tested to see which is more applicable.

A complete overview of the controllers and their interactions is available in Figure 6.6, while the communication between the IMC messages is shown in Figure 6.7.

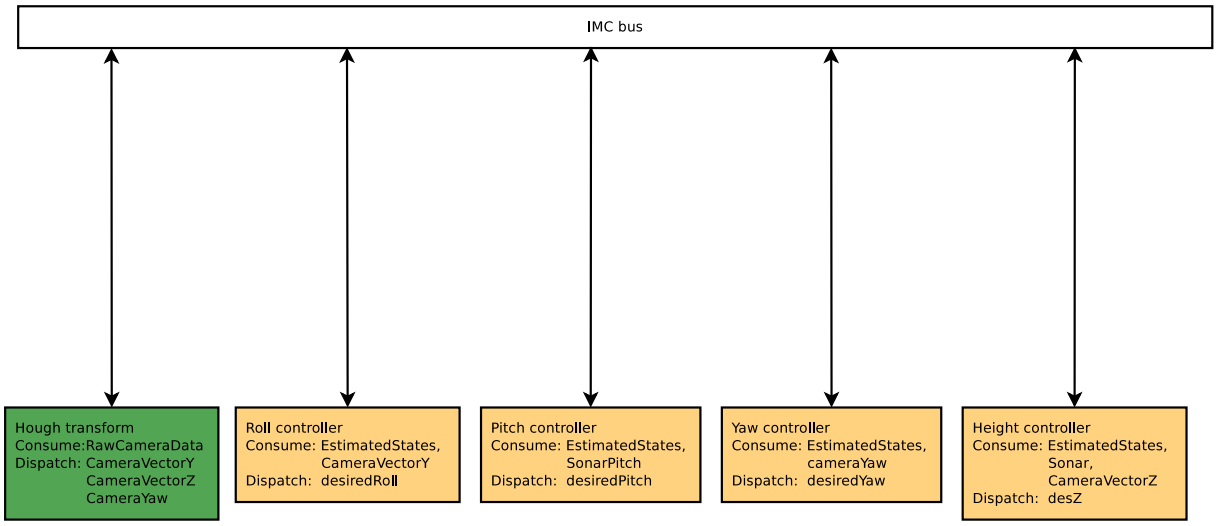


Figure 6.7: Overview of the IMC messages and the functions where they are used. The computer vision task is shown in green and is not covered in this thesis

Chapter 7

Simulation

In the first section of this chapter, the function for finding the origin, radius and angular velocity is verified using a simulated optical flow output with a stationary camera, i.e. $\mathbf{v} = \mathbf{v}_O$. In the next section, two different scenarios are investigated to see how the EKF is able to estimate the metric velocities and distance to a moving target. These scenarios will deal with a stationary and moving target. Furthermore, the simulations of the wind turbine and building inspection scenarios will be presented. All the simulations in the following sections are done in MATLAB R2013a on a Windows computer.

7.1 Estimate angular velocity

In section 4.1.3, a method for estimating the angular velocity based on optical flow measurements was presented. In this section the method is tested using simulated optical flow measurements, see Figure 4.8a. As the figure shows, the algorithm tracked two features on a rotating object for one full revolution. The position of the two feature points and the added optical flow positions, $(x_i, y_i),\ i = 1, 2, 3, 4$, were used to create two lines intersecting in the origin. The radii are estimated by applying Pythagoras. Furthermore, the angular velocity is calculated according to (4.5).

Algorithm 4.2 was simulated with angular velocity of the object, $\omega = 0.2$rad/s, and with the two feature points located at 5 and 10 meters from the center of rotation, respectively. The perpendicular velocity was simulated using (4.5) with added noise

$$\mathbf{v} = r\boldsymbol{\omega} + n_v$$

where n_v is white noise with variance σ^2. The result of the simulation can be seen in Table 7.1 The table shows that the algorithm is able to estimate the angular velocity

Table 7.1: Estimated angular velocity

	Added noise $[\sigma]$			
	0	0.01	0.1	1
Angular velocity of feature 1, ω_1 [rad/s]	0.2	0.1990	0.1953	0.1378
Angular velocity of feature 2, ω_1 [rad/s]	0.2	0.1986	0.1820	0.1323
Estimated angular velocity, $\omega = (\omega_1 + \omega_2)/2$ [rad/s]	0.2	0.1988	0.1887	0.1351

despite of noisy measurements. Even when the noise is 50% of the velocity vector, the algorithm is still able to estimate the angular velocity within some margin. For this reason, the controller will be able to use the estimates to improve tracking.

7.2 Extended Kalman filter

In this section, two scenarios are examined to verify the system model and estimator designed in Chapter 3 and 5. In the following scenarios, the object's global position is fixed, e.g. a wind turbine. Thus, the change of estimated distance is due to movement of the UAV. The code is written and simulated in MATLAB 2013a, and is available in the digital appendix.

7.2.1 Scenario 1: Constant distance from target

In the first scenario, the UAV is moving in the xy-plane (represented in {c}) with a constant distance to the target. The goal of this test is to estimate the metric velocity from the scaled velocity measurement provided by optical flow. The distance is kept constant in order to only estimate the velocity, thus making it easier to verify the simulations.

The test was conducted as follows

- $0 \leq t < 50$ Hover at constant height
- $50 \leq t < 100$ Fly west with constant velocity and $\boldsymbol{\Theta} = \begin{bmatrix} -10^\circ & 0^\circ & 0^\circ \end{bmatrix}^T$
- $100 \leq t < 150$ Hover at constant height
- $150 \leq t < 200$ Fly up with constant velocity and $\boldsymbol{\Theta} = \begin{bmatrix} 0^\circ & 0^\circ & 0^\circ \end{bmatrix}^T$
- $200 \leq t < 250$ Hover at constant height

The result of the simulation is shown in Figure 7.1

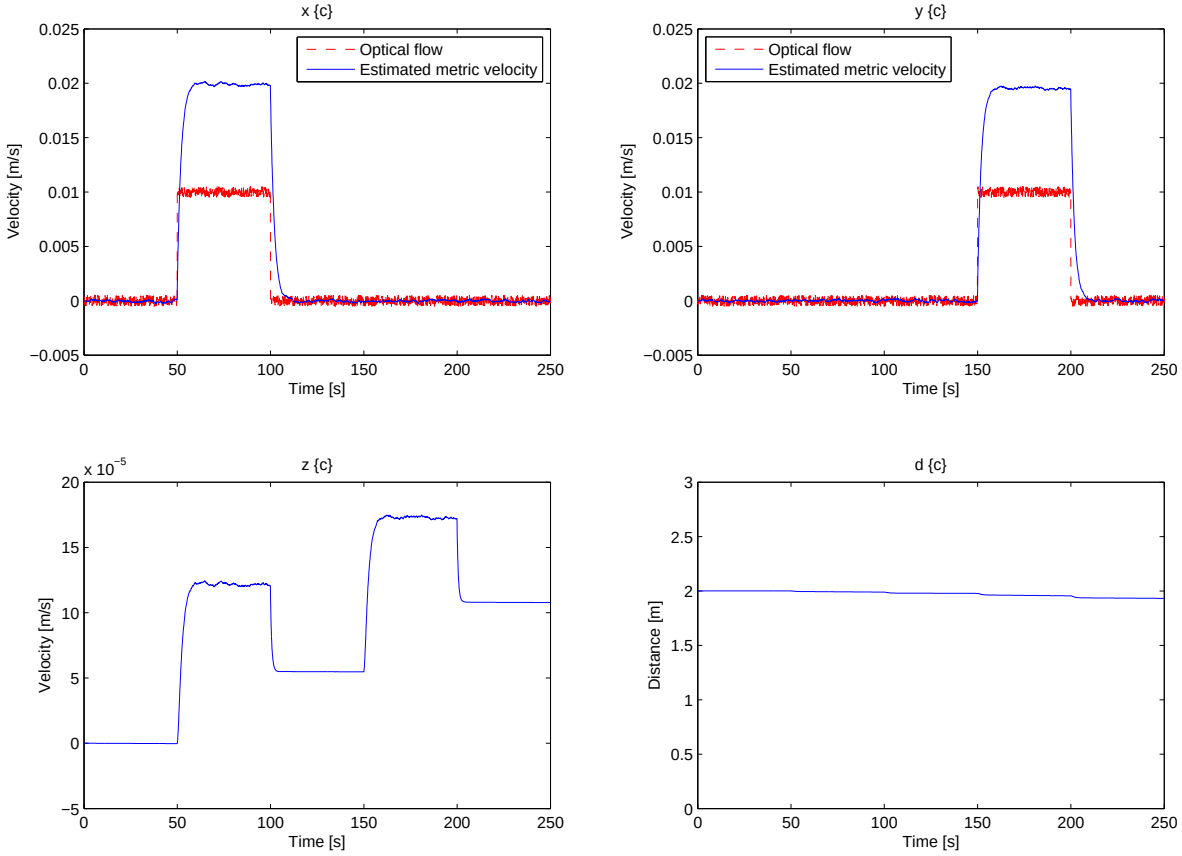


Figure 7.1: Estimating the metric velocity and distance when the UAV is moving with a constant distance to the object

From the figure we see that the estimated metric velocity in x and y-direction are twice as high as the measured optical flow. This is expected as the metric velocities are given as $\mathbf{v} = d \cdot F_{tra}$, and we can see from the figure that the distance is held at approximately 2 meters. The transient effects on the velocity along the z-axis is a result of noisy IMU measurements. As a result, the distance estimate is not completely constant. It is important to mention that in this scenario the dynamics of an UAV and the object is not taken into consideration. For this reason, the received scaled velocities is feasible. In reality, an UAV would not be able to accelerate so quickly.

7.2.2 Scenario 2: Approach object

The second scenario addresses the case were an UAV is approaching the object with constant acceleration. In this test there will be no measured optical flow, so the estimated metric velocity will only depend on the accelerometer measurements. The goal of this test is to see how the observer is able to estimate relative velocity and distance to the target when no observable velocity is available from the camera.

The test was conducted as follows

- $0 \leq t < 50$ Hover at constant height
- $50 \leq t < 100$ Fly towards the object with constant acceleration and $\boldsymbol{\Theta} = \begin{bmatrix} 0° & -10° & 0° \end{bmatrix}^T$
- $100 \leq t < 150$ Hover at constant height
- $150 \leq t < 200$ Fly away from the object with constant acceleration and $\boldsymbol{\Theta} = \begin{bmatrix} 0° & 10° & 0° \end{bmatrix}^T$
- $200 \leq t < 250$ Hover at constant height

The result of the simulation is shown in Figure 7.2

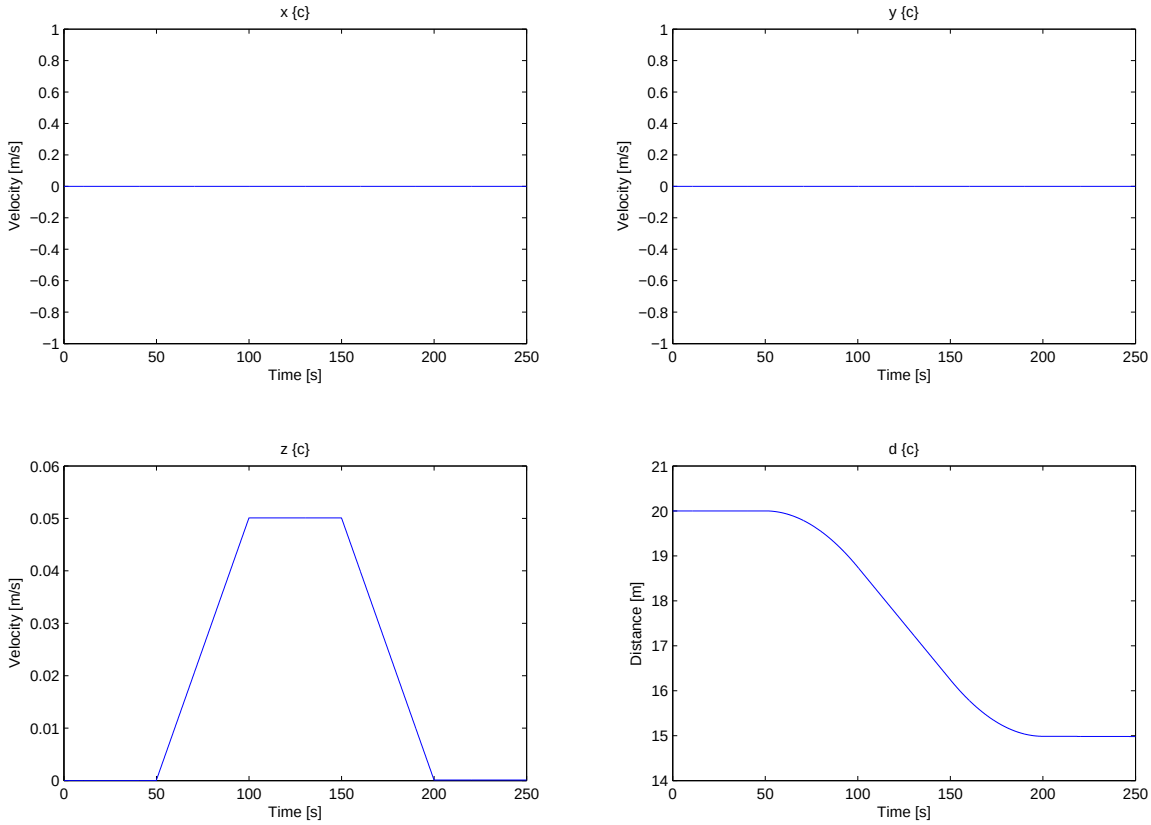


Figure 7.2: Estimating the metric velocity and distance when the UAV is approaching a stationary object with constant acceleration

As expected with constant acceleration, the velocity in z-direction increased linearly. After $100s$ the UAV stopped accelerating and held a constant velocity of $0.05m/s$ towards the target. After approaching the object with constant velocity for $50s$, the UAV deaccelerated until it stopped. The figure shows that without any information from an optical flow algorithm, the distance and velocity estimation in z-direction is only a result of integrating the accelerometer readings.

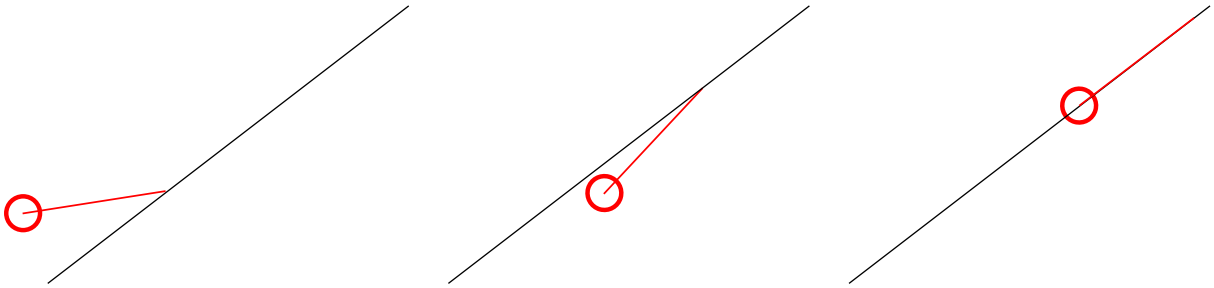


Figure 7.3: Simulated LOS vector received from a camera. The red circle is the UAV and the red line is the LOS vector. The black line represent the wind turbine blade

7.3 Wind turbine

This section addresses the simulation of inspecting a wind turbine. Due to speed at the tip[1] and turbulence, the UAV will not be able to inspect a rotating wind turbine. As a consequence, the following simulation will be of a stationary wind turbine blade. The first step is to simulate the velocity vector received from the camera. The navigation is divided into four parts. The first part examines the navigation to the hub, while the second pans the front of the blade at an arbitrary angle. The third part navigates around the tip of the blade to the back of the blade. Lastly, the different parts are combined to show a complete inspection of a wind turbine blade.

All the simulations in this section are based on the quadcopter UAV simulator created by Peter Corke [11]. The simulations were conducted without any external forces, e.g. wind and measurement noise. The controllers are not tuned perfectly and will only work as a proof of concept. The controllers need to be tuned individually for the hexacopter.

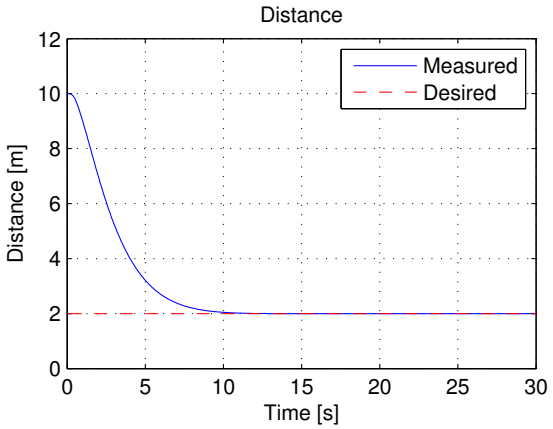
7.3.1 Camera vector

In the simulations, the camera information will not be available. Thus, the information received from a camera needs to be simulated. The camera vector is simulated as a LOS vector from the UAV to the object with a predetermined distance. Since the simulator has the exact GPS position of the UAV and the position of the object is known, the vector can easily be created. In Figure 7.3 the LOS vector is shown for three different UAV positions. The length of the LOS vector can be set in the script to control the speed of the UAV.

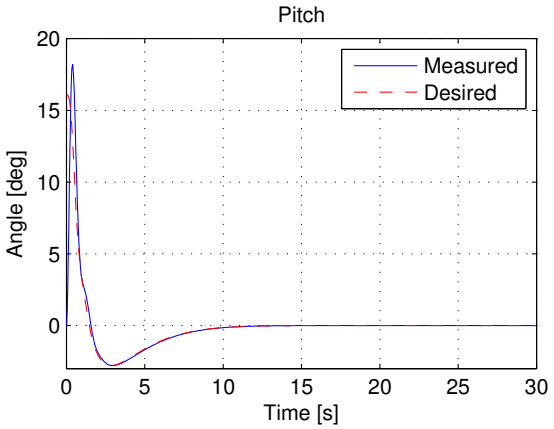
7.3.2 To the hub

In the wind turbine scenario, the first obstacle is to maneuver to the hub. In the following simulation, the UAV starts on the ground 10 meters from the wind turbine. The result of the simulation can be seen in Figure 7.4. From the figures we see that the UAV successfully flies to the hub of the wind turbine. Both the distance and height controllers are designed so that there is no overshoot, see Figure 7.4a and 7.4c. Closer inspection of the pitch angle reveals that the UAV pitches forward in order to approach the hub. After a couple

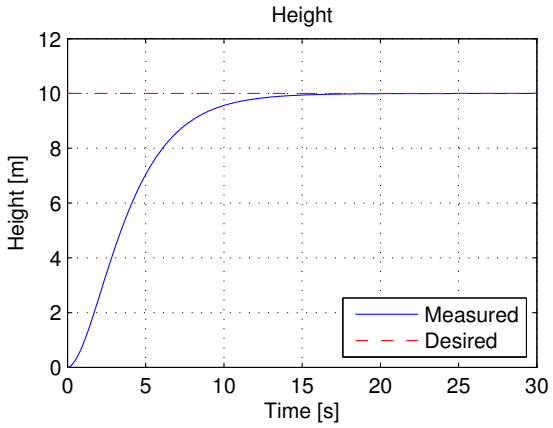
[1]Example: A wind turbine with 40 meters blades rotating at 10 rpm will have a speed of 150 km/h at the tip



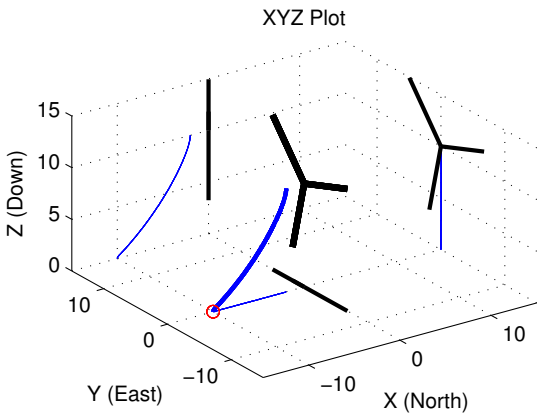
(a) Measured vs desired distance



(b) Measured vs desired pitch



(c) Measured vs desired height



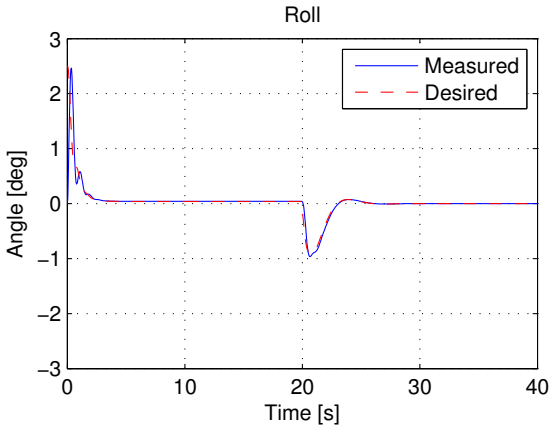
(d) Three-dimensional plot of the UAV and wind turbine. The red circle marks the starting point.

Figure 7.4: Plot of the distance, pitch, height and the three-dimensional trajectory as the UAV approaching the wind turbine

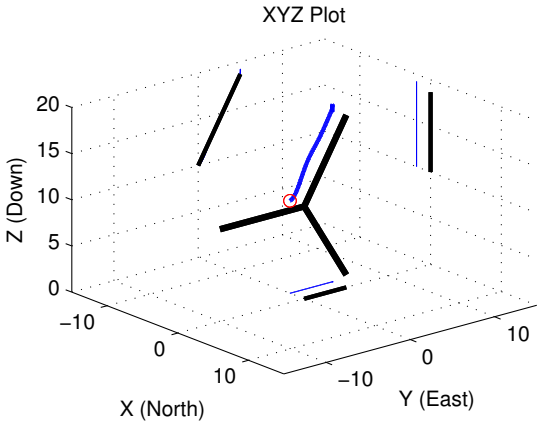
of seconds, the UAV has gains great speed forward and starts pitching the opposite way to cancel its momentum forward. This leads to a stop at the correct distance from the hub, in this case 2 meters. In Figure 7.4d, the blue line shows the trajectory of the UAV with a starting point at the red circle, while the wind turbine is illustrated with black lines 120° apart.

7.3.3 Pan the blade

After the UAV reaches the hub of the wind turbine, the next step is to inspect the front of the blade for damages. In the simulations, the angle of the blade can be set arbitrary from 0° to 360°. In this scenario, the angle of the blade is set to 60°. The result of the simulation can be seen in Figure 7.5. The UAV is capable of following the desired roll angle with satisfying results, see Figure 7.5a. In the same way as the UAV pitched the opposite way to stop the momentum forward, it is clear from the figure that the UAV uses the same technique with roll at the tip of the blade. The three dimensional plot, Figure 7.5b, clearly shows that the UAV follows the blade and stops at the end.



(a) Measured vs desired roll



(b) Three-dimensional plot of the UAV and wind turbine. The red circle marks the starting point.

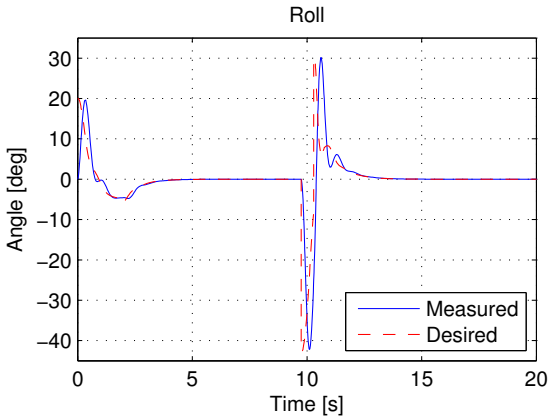
Figure 7.5: Plot of the roll and the three-dimensional trajectory as the UAV inspecting a wind turbine blade

7.3.4 Around the tip

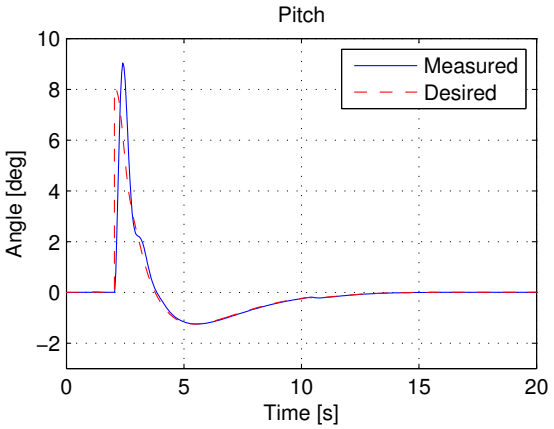
In order to inspect the back of the blade as well, the UAV needs to maneuver around the tip of the blade. In the following simulation the UAV starts at the tip of the blade and flies around in a square with the same distance as the starting point, 2 meters in this case. When it reaches the back, and the blade is within sight, the UAV stops and hovers, ready to inspect the back of the blade. The result of the simulation can be seen in Figure 7.6. From the figure we see that after about 10 seconds, right after the UAV reaches the back, the desired roll angle experiences some glitches. This is due to the necessity of simulating the velocity vectors usually provided by a camera. Similarly to the two other experiments, the roll and pitch angles produces negative desired angles to stop the momentum sideways and forward, respectively. Figure 7.6c shows the distance from the UAV to the blade. When the UAV is position behind the blade, the sonar measures negative distance. This is to simplify the simulation of the sonar readings. In the last figure, Figure 7.6d, we see that the UAV accomplish the task of navigating around the tip.

7.3.5 The blade

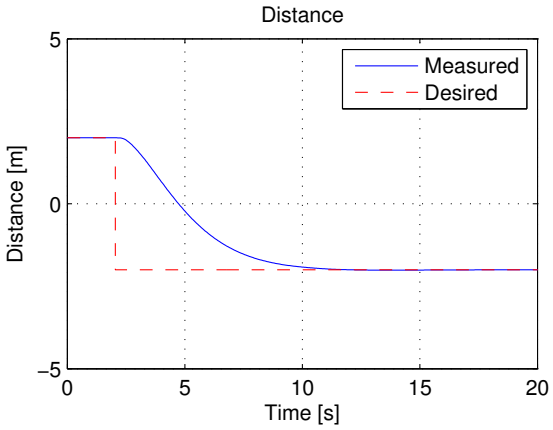
By connecting the different scenarios described earlier, the UAV can now inspect the front of the blade, fly around the tip and inspect the back of the blade. Inspection of the back of the blade is accomplish by reversing the velocity vector used on the front. The result of the simulation can be seen in Figure 7.7. In Figure 7.7a - c, we see the same phenomenon as in the previous simulations. The last figure, Figure 7.7d, shows how the UAV starts at the hub of the wind turbine, inspects the front, flies around the tip and inspects the back of the blade. After finishing inspecting the back, the UAV hovers at the hub ready to inspect the next blade.



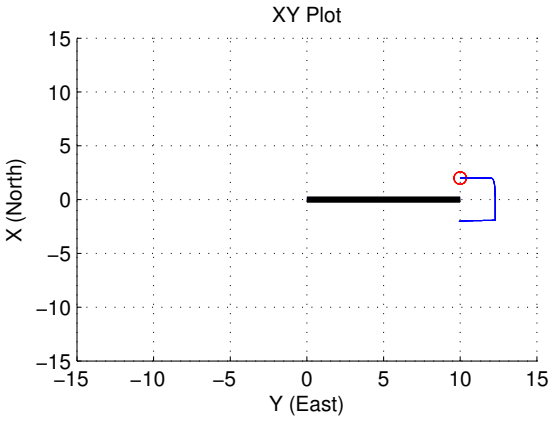
(a) Measured vs desired roll



(b) Measured vs desired pitch

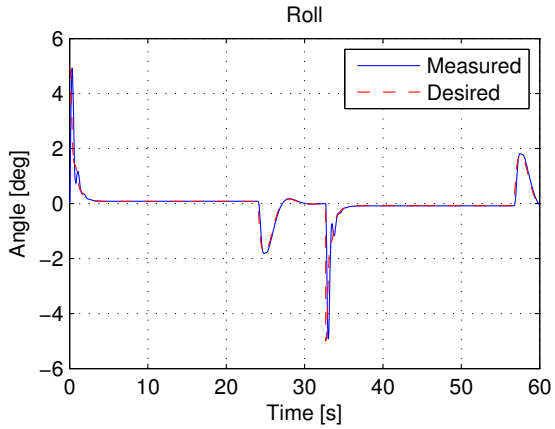


(c) Measured vs desired distance

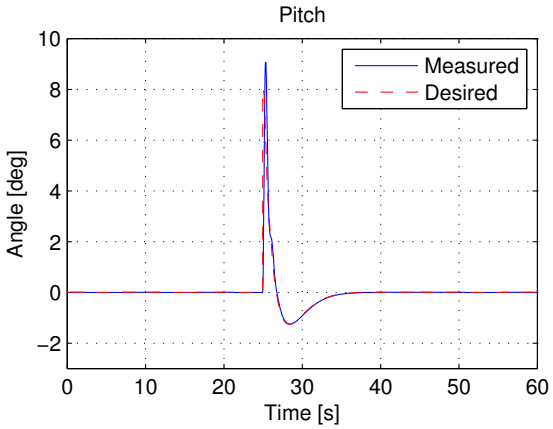


(d) Two-dimensional plot of the UAV and wind turbine. The red circle marks the starting point.

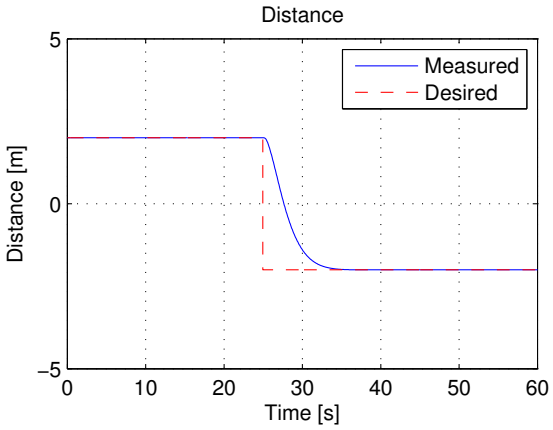
Figure 7.6: Plot of the roll, pitch, distance and the two-dimensional trajectory as the UAV flies around the tip



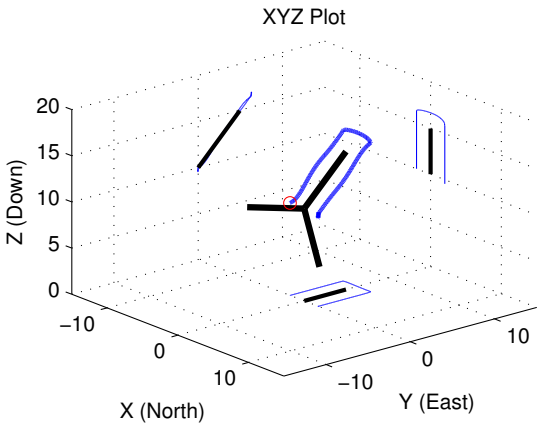
(a) Measured vs desired roll



(b) Measured vs desired pitch



(c) Measured vs desired distance



(d) Three-dimensional plot of the UAV and wind turbine. The red circle marks the starting point

Figure 7.7: Plot of the roll, pitch, distance and the three-dimensional trajectory as the UAV inspect the blade

7.4 Building inspection

This section addresses the simulation of inspecting a building for structural damages. The simulation of building inspections has significant similarities to the wind turbine simulation, as both receive desired velocity vectors from the camera. There are several different search patterns capable of inspecting a building, e.g. parallel, creeping line, square, sector [8]. This thesis will consider the two most popular patterns, parallel and creeping line search.

7.4.1 Parallel search

In parallel search the UAV inspects the building one vertical segment at the time. Dependent on the FOV of the camera and distance to the building, the spacing between the search lines changes. Instead of manually calculating the desired spacing between the search lines, it can be calculated using information from the camera and sonar. The desired spacing can be calculated as

$$y_{desired} = 2d \tan\left(\frac{\alpha}{2}\right) \tag{7.1}$$

where d is the distance to the building, provided by the sonar, and α is the FOV of the camera in radians, see Figure 7.8. By using (7.1) the algorithm will not be dependent on the choice of camera nor the distance to the building. Consequently, the UAV can fly closer to the building in selected areas where closer inspection is necessary.

Figure 7.9 shows the simulation of an UAV inspecting a building for structural damages using the parallel search method. In this simulation the height of the building is set to 20 meters and the distance from the UAV to the building is set to 1 meter. By using a camera with a FOV of 90° ($\pi/2$ rad), the horizontal distance between the search lines is 2 meters.

In Figure 7.10, a simulation of an UAV inspecting a building with close inspection in a region is shown. The figure shows that the UAV flies closer to the building to inspect for damages in more detail. As a result, the horizontal distance decreases from 2 meter to 1, according to (7.1). After the UAV is done with the close inspection, the distance to the building again increases to 1 meter. Consequently, the horizontal distance increases to 2 meters.

7.4.2 Creeping line search

In some cases there may be desirable to inspect the building horizontally instead of vertically, dependent on the structural damages and shape of the building. This is achieved by using creeping line search. The technique described in (7.1) can also be used to make the inspection more adaptive. Figure 7.11 shows how the UAV can be used in building inspections by using creping line search.

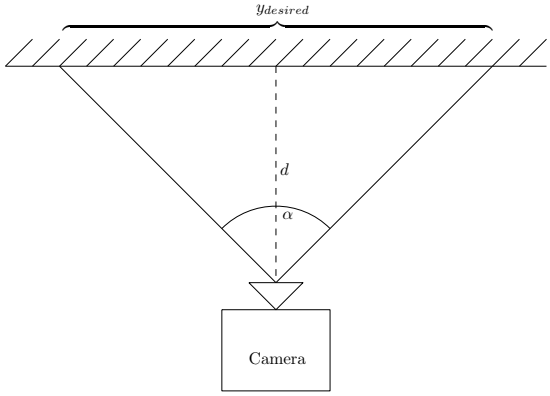


Figure 7.8: The spacing between the search lines is a function of the distance to the building and the FOV of the camera

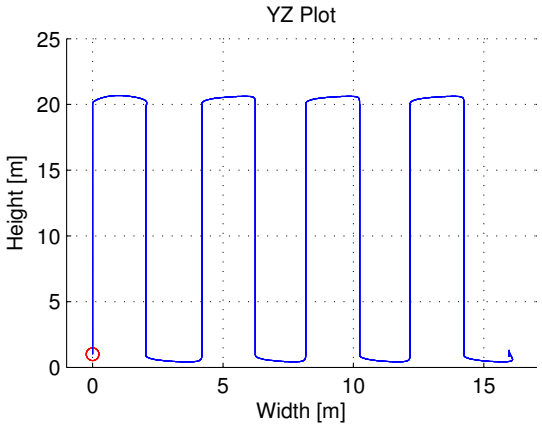
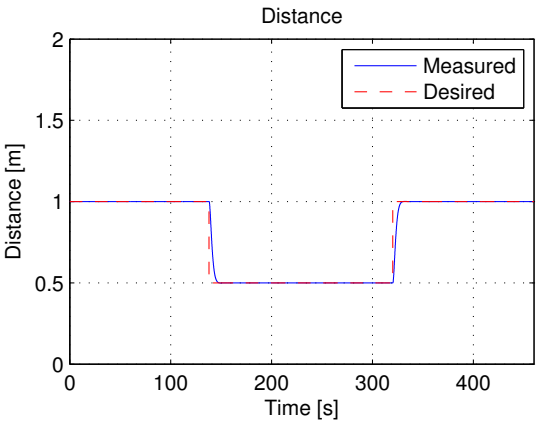
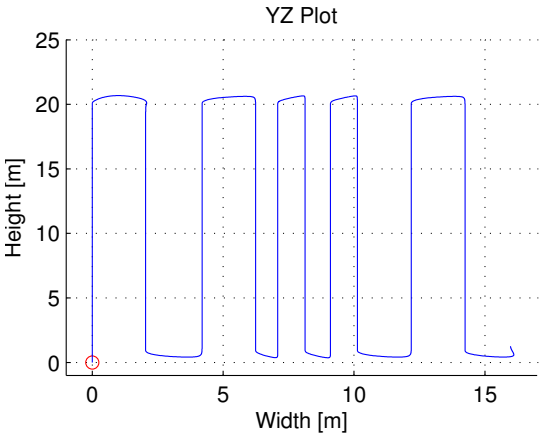


Figure 7.9: YZ plot of the UAV inspecting the building using parallel search



(a) Measured vs desired distance



(b) YZ plot

Figure 7.10: Two-dimensional plot of the UAV inspecting a building with closer inspection

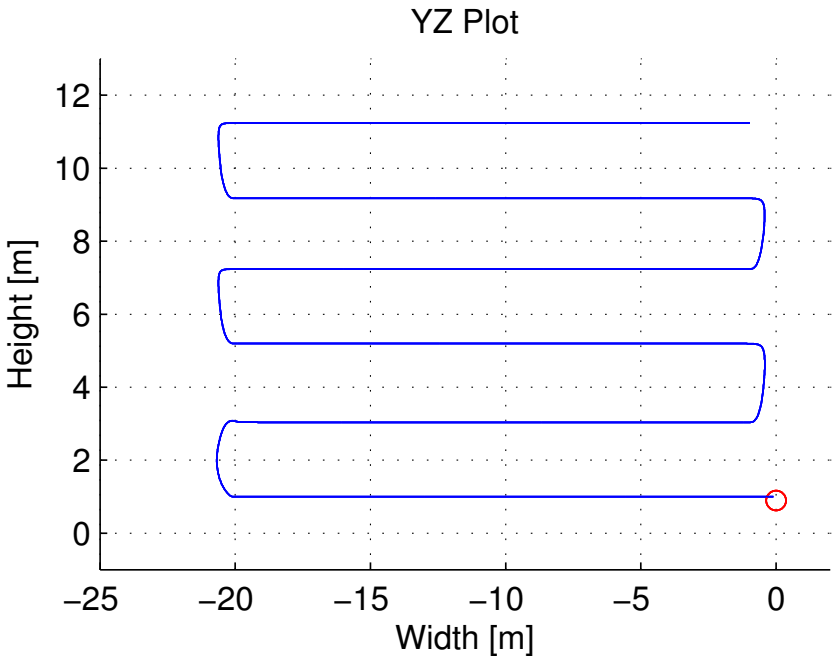


Figure 7.11: Plot of the UAV inspecting a building using creeping line search

7.5 Location of the UAV with respect to the object

In some cases, it may be beneficial for the operator to know where the UAV is relative to the blade or building. This makes the operator able to locate the damage, and thus send repairmen to the correct location. As previously mentioned, a wind turbine blade has a lack of features to track and consequently one can not take advantage of optical flow algorithms. In order to estimate where the UAV is in relation to the blade, a few approaches can be utilized. Firstly, the accelerometer readings can be used to estimate the distance covered by the UAV from the hub. However, the estimates will be highly influenced by bias and noise, and will drift due to integration, see Appendix A. Secondly, it is possible to use a priori information about the blade dimension and shape, e.g. width, length and angle. By using the output of the Hough transform together with the distance estimate from the sonar, it is possible to estimate the position of the UAV with respect to the blade. This requires that the thickness of the blade changes according to the position on the blade, i.e. that every measurement equals a unique position.

In the case of building inspections, there are features to track and other approaches can be used. The optical flow algorithm will give a relative velocity estimate between the UAV and the blade. Dependent on the amount of noise, the velocity estimate can be integrated to give an approximation of the distance travelled. If it turns out that the velocity estimate is too noisy, it is also possible to use a priori information about the building, e.g. number of windows, to correct the estimate. However, this method would have to be adjusted for each individual building as the number of windows and the spacing between them changes.

Chapter 8

Software In The Loop Simulation

Software-in-the-loop (SITL) simualtion virtualize the ArduPilot, UAV and environment with a simulator. The simulator features a dynamic model of the UAV and the environment. The SITL simulation is conducted to test the software that will be executed on the UAV. Consequently, the communication and interactions between tasks over the IMC bus is tested, along with information about run-time and stability.

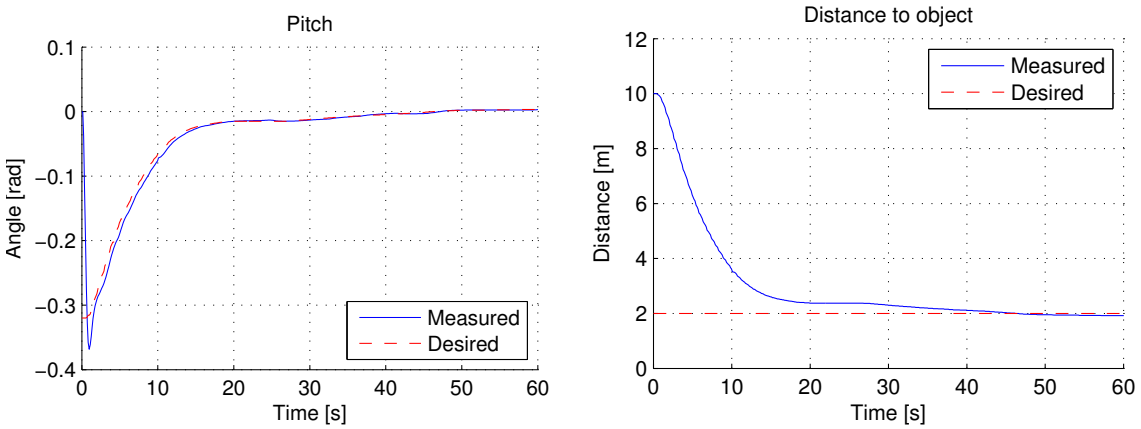
This chapter will take a closer look at the SITL simulation with external forces. The external forces are environmental forces, e.g. wind, and sensor errors, e.g. bias, drift and accuracy of the GPS. The first section tests the pitch controller independently. The second section connects the height, pitch and roll controller together in the scenario of inspecting the front of a wind turbine blade. Lastly, a discussion of the results is made.

8.1 Pitch

In the SITL simulations, the position of the UAV is simulated perfectly. Consequently, the sonar measurements can be simulated in the same manner as in section 7.3.2, i.e. subtract the position of the UAV and a predefined object. In the simulation, the UAV started 10 meters from the object and flew closer until it was 2 meters away. The height was kept constant at 10 meters. The result of the simulation can be seen in Figure 8.1. The figure shows that the UAV is capable of following the desired pitch angle with satisfying results. Similar to the simulation done in the previous chapter, the UAV successfully maneuvers to the desired distance of 2 meters. However, the UAV used considerable more time than previously. This can be improved be tuning the PD parameters more carefully.

8.2 Velocity vector

Similarly to section 7.3.1, the camera vector was simulated using a LOS vector. Furthermore, the vector was decomposed in y and z-direction and sent to their respective controller. Figure 8.2 shows the result of the simulation. The first figure shows how the measured roll angle follows the desired angle. As oppose to the pitch controller in the previous section, the roll controller does not track the desired angle perfectly. The second figure shows how the pitch controller does small corrections to stay at a constant distance to the object. The desired height is plotted against the measured height in the third



(a) Measured vs desired pitch

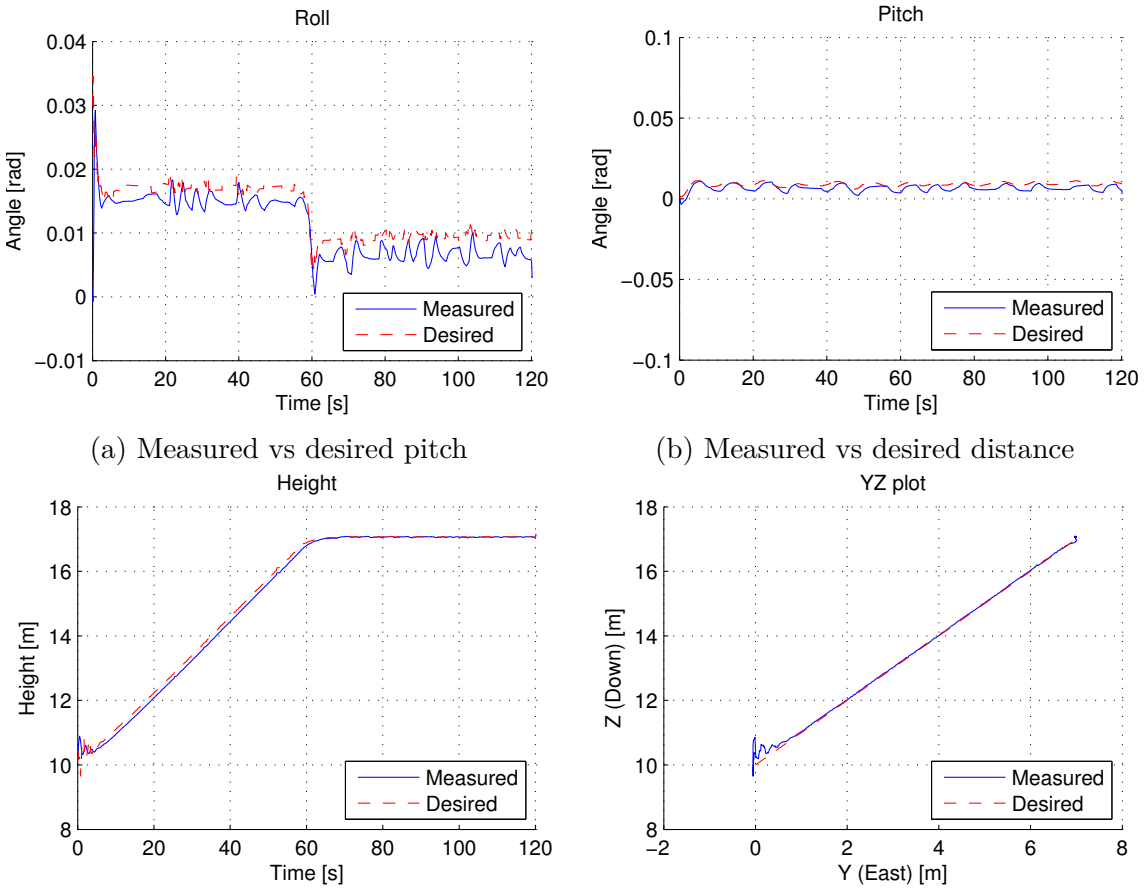
(b) Measured vs desired distance

Figure 8.1: Plot of the pitch and distance as the UAV approaching an object

figure. The height controller experiences small deviations in the measured height due to a dynamic desired height. This can be improved by fine-tuning the control parameters. The two-dimension YZ plot is shown in the last figure. Similarly to the simulations in the previous chapter, the UAV follows the desired path with high accuracy. There are a some transients at the beginning due to the necessity of manually switching into DUNE mode.

8.3 Discussion

This section has presented the SITL simulations of keeping a constant distance and follow a velocity vector. The simulations have proven to be less accurate than the simulations done in the previous chapter. This is due to errors on the UAV model, lower update frequency and roughly tuned parameters. In addition, the necessity of manually switching into autonomous mode, resulted in different starting points for each simulation. Thus further complicated the tuning.



(a) Measured vs desired pitch

(b) Measured vs desired distance

(c) Measured vs desired height

(d) YZ plot

Figure 8.2: Plot of the roll, pitch, height and the two-dimensional trajectory as the UAV inspecting a wind turbine blade

Chapter 9

Hardware testing

This chapter addresses the testing of the external components used in this thesis. The first section will compare the sonar with different connections. The power supply will also be tested together with an endurance test of the battery. These two sections will use a Arduino[1] micro controller to log the distance estimates and cell voltage of the battery. The last section addresses the test of the different controllers on the UAV platform and presents the results.

9.1 Maxsonar

In this section the MaxBotix MaxSonar EZ4 is tested with different connections. The experiments were conducted using analog, PWM and serial communication. Furthermore, by mounting the sonar between two legs, the sonar will not experience any reflections. This is due to the narrow beam width of the chosen sonar. The experiments were conducted using a Leica DISTO D3[2] to accurately measure the distance.

The first section addresses the use of analog, while the second take a closer look at PWM. Finally, the serial communication is tested together with a discussion and a conclusion of the different connections.

9.1.1 Analog

The sonar was connected as seen in Figure 9.1 and the estimated distance was logged to a file. The RC filter is included to reduce the amount of electrical noise. The result of the experiment is available in Table 9.1. The table shows that the sonar is able to estimate the distances with a accuracy of 2-6 centimetre regardless of the distance.

The main disadvantage of using analog is the accuracy. Analog is the least accurate of the available connections. On the other hand, the sonar is easily connected to the APM trough one of the analog inputs.

[1] Open-source electronics prototyping platform. See `www.arduino.cc/` for more information

[2] Laser range finder with accuracy of 1mm and range of 100m

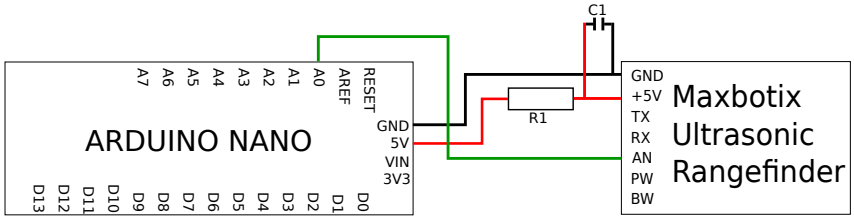


Figure 9.1: Schematic of the Arduino connected to the MaxSonar EZ4 sonar using analog

Table 9.1: Sonar measurement using analog connection

MaxSonar EZ4 [cm]	Leica DISTO D3 [cm]	Error (absolute value) [cm]	Error [%]
50	55	5	10
100	104.5	4.5	4.5
150	154.8	4.8	3.2
200	204.8	4.8	2.4
250	254.1	4.1	1.64
300	305.8	5.8	1.93
350	352.9	2.9	0.83
400	403.2	3.2	0.8
450	454.3	4.3	0.96
500	505.7	5.7	1.14
550	554.5	4.5	0.82
600	603	3	0.5
650	652.6	2.6	0.4
700	704.8	4.8	0.69
750	755.1	5.1	0.68

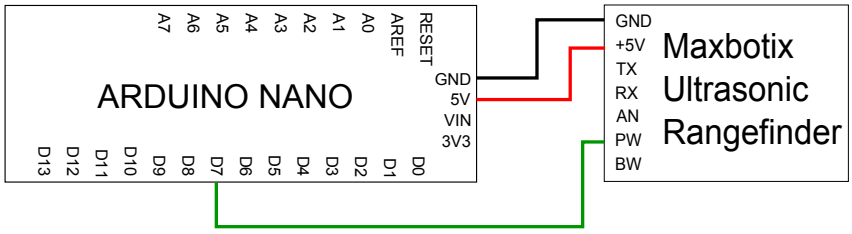


Figure 9.2: Schematic of the Arduino connected to the MaxSonar EZ4 sonar using PWM

Table 9.2: Sonar measurement using PWM connection

MaxSonar EZ4 [cm]	Leica DISTO D3 [cm]	Error (absolute value) [cm]	Error [%]
50	52.1	2.1	4.2
100	102.0	2.0	2.0
150	152.4	2.4	1.6
200	201.7	1.7	0.85
250	252.7	2.7	1.08
300	301.7	1.7	0.57
350	352.8	2.8	0.80
400	402.2	2.2	0.55
450	451.1	1.1	0.24
500	501.4	1.4	0.28
550	550.5	0.5	0.09
600	600.6	0.6	0.10
650	649.9	0.1	0.02
700	700.7	0.7	0.10
750	749.4	0.6	0.08

9.1.2 PWM

In the case of connecting the sonar through PWM, the sonar was connected to one of the digital pins on the Arduino, see Figure 9.2. The distance estimate was logged and the result of the experiment can be seen in Table 9.2. Unlike the test in the previous section, the sonar is able to estimate the distance within 1-2 centimetres precision regardless the distance. The table also reveals higher precision with increased distance. This is due to the internal clock on the Arduino. At small distances the time between the sound wave and the echo will be equally short. Hence, it's easier to measure the time precisely when the distance is greater.

The main disadvantage of using PWM in this project is the lack of PWM support on the Pandaboard. One solution could be to add an external circuit to convert the PWM signal to serial. The update frequency is also lower (≈10Hz) due to the necessity of filter the measurements using a mode filter.

9.1.3 Serial

The last of the available connections is serial, as shown in Figure 9.3. Since the sonar only transmits, and not receive, only the transmission (TX) pin is required. The result of the

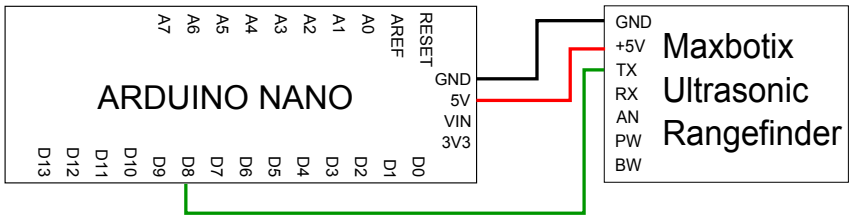


Figure 9.3: Schematic of the Arduino connected to the MaxSonar EZ4 sonar using serial communication

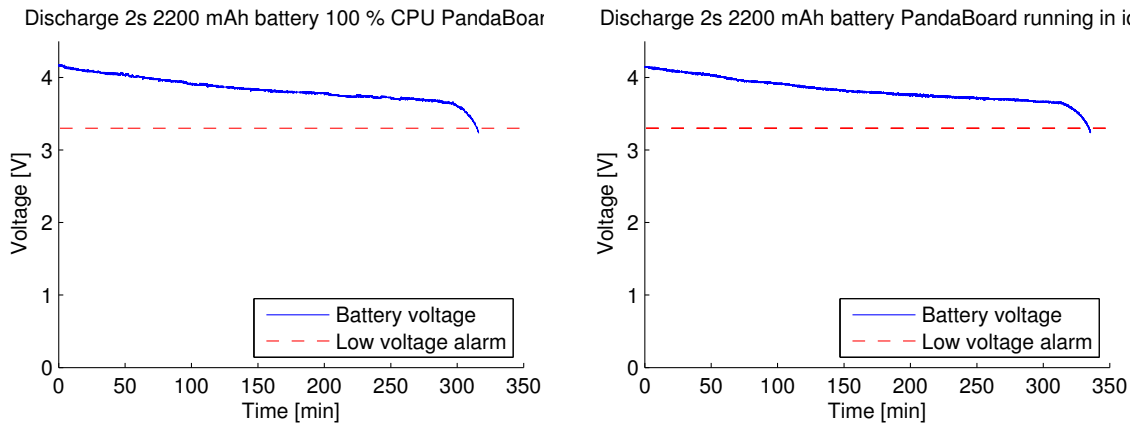
Table 9.3: Sonar measurement using serial connection

MaxSonar EZ4 [cm]	Leica DISTO D3 [cm]	Error (absolute value) [cm]	Error [%]
50	53.5	3.5	7.00
100	102.6	2.6	2.60
150	152.1	2.1	1.40
200	202.0	2.0	1.00
250	251.8	1.8	0.72
300	301.6	1.6	0.53
350	351.0	1.0	0.29
400	401.1	1.1	0.28
450	450.8	0.8	0.18
500	500.2	0.2	0.04
550	548.7	1.3	0.24
Out of bounds	-	-	-

experiment is shown in Table 9.3. By comparing the three tables, we see that precision wise, the serial is on the same level as PWM. However, the maximum distance decreased significantly. The maximum distance was 5.5 metres contrary to the 7.5 metres achieved with analog and PWM. The serial communication can be connected to the PandaBoards GPIO and will deliver a notably higher update frequency (≈50Hz).

9.1.4 Discussion

This section has shown that there are some differences between the three interfaces when it comes to precision. The analog interface offered the least precise estimates, but is easily connected to the APM and DUNE. PWM was highly accurate and was able to measure the maximum distance of 7.5 meters. However, in order to connect the sonar via PWM to the PandaBoard, an external circuit is required due to lack of support of PWM on the PandaBoard. The sonar can be connected to one of the available digital inputs on the APM, but this will require significant modification of the code. In the case of serial connection, the precision was comparable to the PWM, but it was not able to measure the maximum distance.



(a) 100 % CPU (b) Idle

Figure 9.4: Discharge of a battery powering the PandaBoard at full stress test (a) and idle (b)

9.2 Power supply to Pandaboard

The PandaBoard was tested with a 2200 mAh two cells Turnigy battery and a 3A UBEC. The stress test was conducted using the stress[3] command on the PandaBoard. The stress tool runs the PandaBoard with 100 % CPU and memory usage. A low voltage lipo alarm was attached to prevent any damages on the battery. The voltage of one of the battery cells was logged using an Arduino Nano v3. The result of the test can be seen in Figure 9.4a.

The figure shows that the cell voltage of the battery decreases linearly most of the time. However, when the voltage went below approximately 3.7V, the voltage decreases a lot faster. A lipo alarm will output a high pitch sound when the voltage of one of the cells is lower than 3.3V. The experiment showed that it may be advantageous to give an earlier warning, to be able to abort the mission and return to launch before the voltage drops to under 3.3V. This could prevent any damage to the battery. The battery lasted for roughly five hours.

The test above was conducted to determine the lower bound of the endurance of the battery. However, when testing out in the field, there will be moments where the UAV is not on a mission and not running DUNE. To see if the battery would last a whole day of mixed usage, another test was conducted. In this test, the PandaBoard ran on idle, i.e. DUNE was not running, only the OS. Figure 9.4b shows the result of the test. The figure shows that the endurance of the battery did not increased significantly. Hence, the usage of the PandaBoard does not affect the battery life. Similarly to the full stress test, the battery voltage decreased linearly before the voltage dropped to 3.7V.

[3]Stress test: `http://manpages.ubuntu.com/manpages/lucid/man1/stress.1.html`

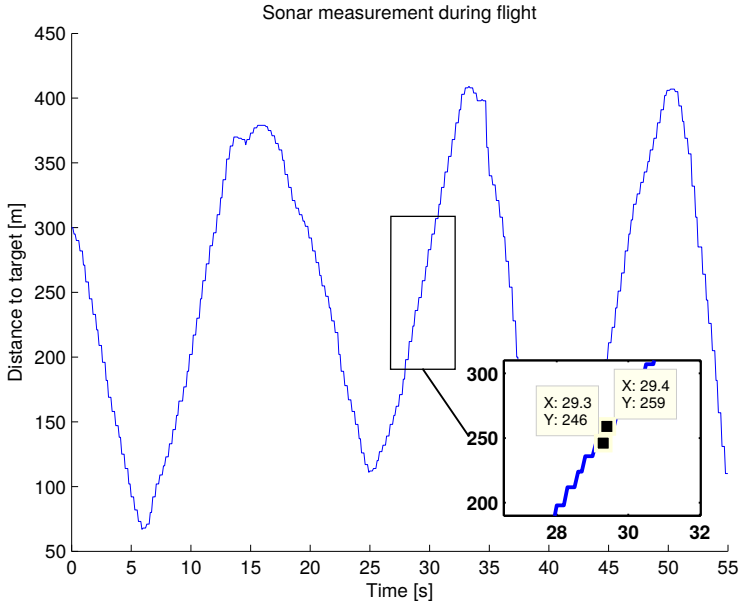


Figure 9.5: Sonar measurement during flight in manual mode

9.3 UAV

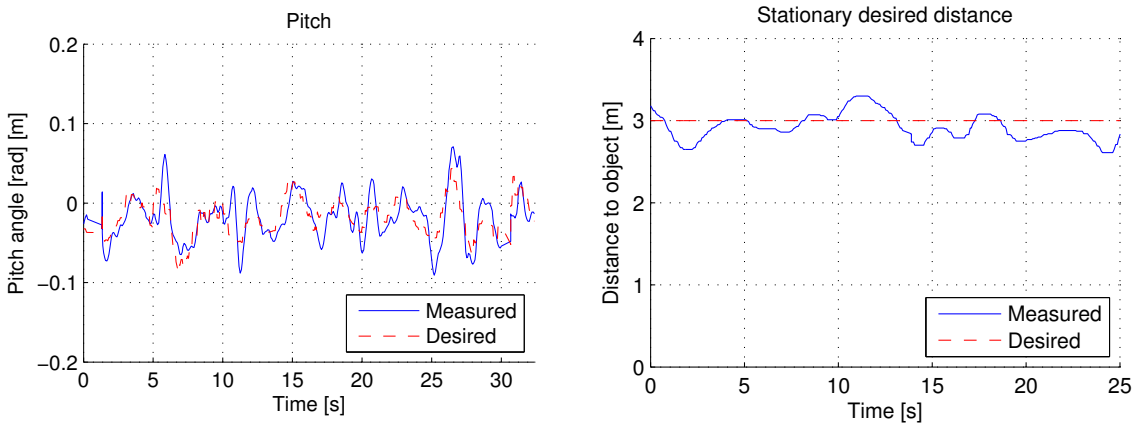
In the following section the result of the test flying is presented. The experiments were performed in a lab equipped with a safety net, see Appendix B. The payload was equipped with a PandaBoard, USB camera and two sonars for measuring the distance to the object and the height above ground, respectively. This section is structured as follows. Firstly, the pitch controller using a front facing sonar is presented. Secondly, the roll controller using camera information was tested with a stationary object. Lastly, a comparison of a down facing sonar and barometer is made before the throttle controller was tested.

The results of the test are shown as videos in the digital appendix.

9.3.1 Pitch controller

As mention in section 2.6.1, the sonar is highly affected by noise and turbulence. In order to examine the influence of noise and turbulence, and to test the stability of the sonar in dynamic applications, the sonar measurements were logged during a manual flight. The result of the experiment is shown in Figure 9.5. The figure shows that the sonar does not experience any spikes or noise in the measurements. However, as shown in the magnified plot, the sonar does not measure one centimetre at the time. The sonar is only able to give measurements in increments of approximately 10cm. This is due to a low frequency on the sonar readings and that the pitching of the UAV causes the sonar to measure diagonally instead of the perpendicular distance.

The pitch controller was tested by manually controlling the throttle, yaw and roll, while the Pandaboard sent the desired pitch angle to the ArduPilot. The desired distance is initialized at startup and can be changed through Neptus during flight. The result of the test can be seen in Figure 9.6. The figure shows that the desired pitch angle is continuous



(a) Measured vs desired pitch

(b) Measured vs desired distance to object

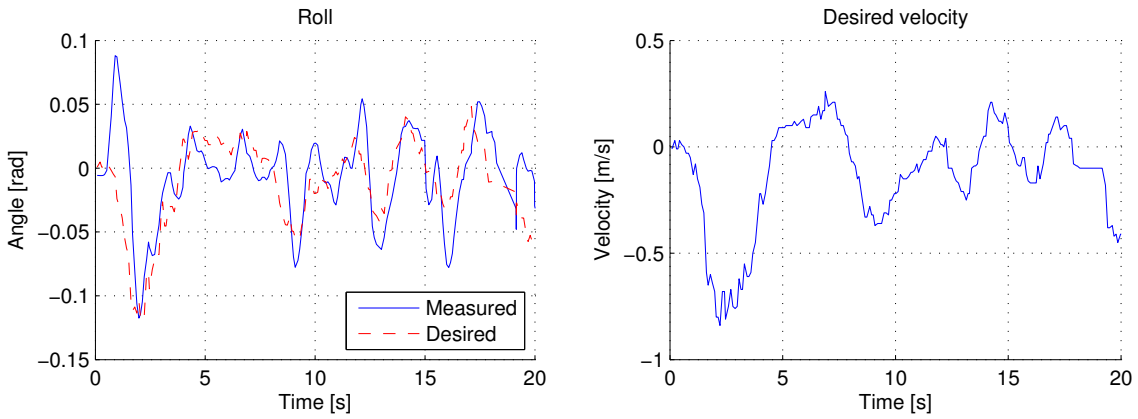
Figure 9.6: Plot of the pitch and distance to an object with constant desired distance

without any noise or spikes. However, the pitch controller built-in the ArduPilot software oscillates around the desired pitch angel, making the UAV unable track the desired angle perfectly. This can be improved by tuning the PID parameters more carefully.

As shown above, the sonar measurements are given in increments. This, in addition to roughly tuned PID parameters, led the UAV to oscillate around the set point. Furthermore, the sonar was mounted directly to the payload. Hence, the distance measured by the sonar will change relative to the pitch angle of the UAV. A solution to this problem is to mount the sonar on a gimbal, which will always control the sonar so that it is perpendicular to the object, or by correcting the reading using the measured pitch angle. Despite these inaccuracies, the UAV stayed within the desired distance with a 30 cm error, which in most cases is satisfactory.

9.3.2 Roll

In section 6.3.2, the roll controller was designed using camera information to calculate the desired roll angle. Figure 9.7 shows the result of an experiment where the UAV is controlled to a stationary object. This replicates the scenario of hovering at the hub ready to inspect the blades. The experiment was conducted to verify that the computer vision algorithm calculates the correct velocity vector, and that the roll controller is able to control the UAV to hover at the same location. The figure shows how the measured roll angle follows the desired angle computed by the controller. Similarly to the pitch controller, the roll controller is roughly tuned and the measured angle does not follow the desired angle perfectly. The velocity vector received from the computer vision algorithm is shown in the figure on the right. The measurements had some noise and the frequency was lower than the control loop. However, the UAV successfully hovered at the hub, as indicated by the desired velocity, i.e. fluctuated around zero. The distance to the object was held constant at approximately 2 meters.



(a) Measured vs desired roll

(b) Desired camera vector

Figure 9.7: Plot of the roll and camera error as the UAV hovering at the hub

9.3.3 Throttle

In order to measure the height above ground, the built-in barometer or an external sonar can be applied. Figure 9.8 shows a comparison of the estimated altitude from the barometer and sonar. The experiment revealed that the barometer does not provide the necessary accuracy at low altitudes (< 0.5m). This is because the propellers generate turbulence near the ground. The sonar has problems at low altitude as well, but this is limited to below 20cm. Apart from this, the barometer and sonar readings are similar within a few centimetres of each other. For the experiments conducted in the net, the sonar was used. However, when testing outside on a wind turbine, a sonar will not be applicable due to its limited range. Further investigation needs to be conducted to see if the barometer has the necessary accuracy at high and windy conditions. The ArduCopter code was changed to allow a downward-facing sonar as height reference instead of the barometer.

To test the performance of the altitude controller, two different experiments were conducted; stationary and time-varying set point. The result of the tests can be seen in Figure 9.9. The figure on the left shows how the UAV responds to a unit step response. The UAV overshoots with approximately 30cm, but stays at the desired altitude with an error of $\pm$5cm. In the figure to the right, the set point was increased/decreased with increments of 3cm as soon as the UAV reached the current set point. This ensures that the climb rate stays low, thus preventing uncontrollable behaviour from the UAV. The figure shows that the UAV is able to track the desired height with satisfactory results. However, at the end points and from a ground start, the UAV overshoots. In the case of a ground start, the sonar is not able to measure lower distances than 20cm and thus the UAV experiences a step response. The overshoots can be reduced by carefully tuning the control parameters of the in-built height controller.

9.3.4 Discussion

In this section, the different controllers were tested on the UAV platform. The tests revealed that the in-built controllers had problems following the desired set points. However, the UAV maneuvered as it was suppose to, despite roughly tuned controllers. The

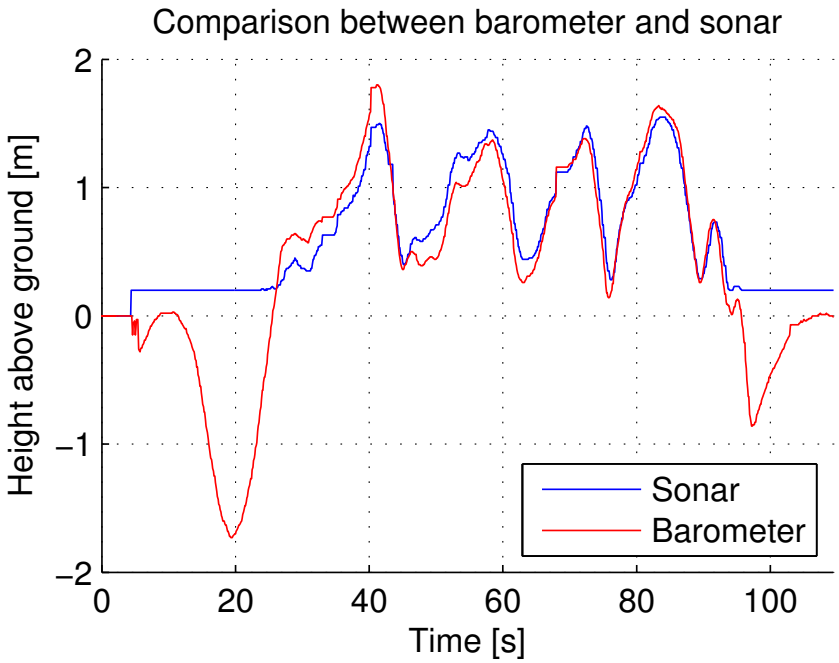
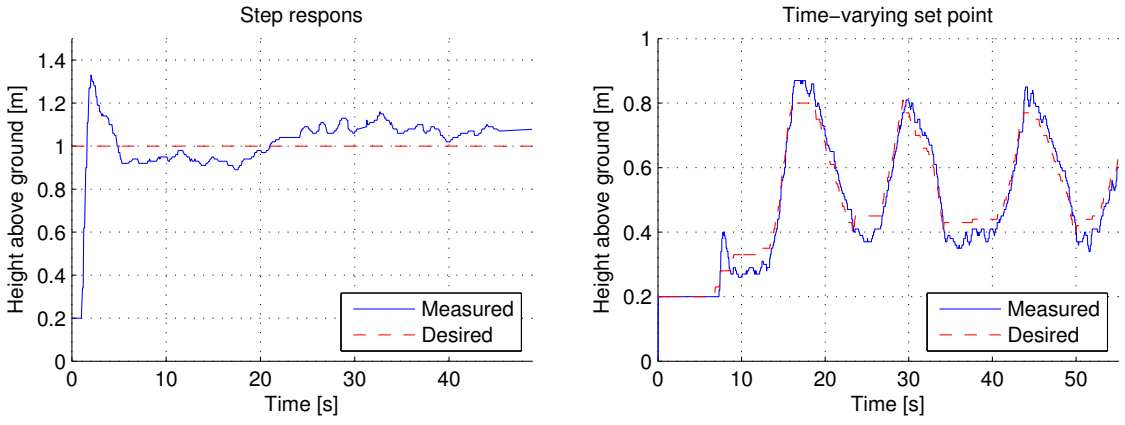


Figure 9.8: Comparison of the estimated relative height using a barometer (blue) and sonar (red)



(a) Stationary set point

(b) Time-varying set point

Figure 9.9: Measured vs desired height with step response and time-varying set point

results will serve as a proof of concept and both the in-built controllers and the controllers used in this section can be tuned and achieve significantly better results. In addition, the camera information utilized in this thesis was a bit noisy, which affected the stability of the roll controller. The distance and height controllers can be modified to be more precise by using a LRF. The LRF will also increase the range of the distance estimate, which is necessary in the case of wind turbine and building inspection.

Chapter 10

Conclusion and future work

The following chapter gives a conclusion of the work and results presented in this thesis. Moreover, the future work necessary for implementing the observer on the UAV is presented. The section also gives ideas for further improvements and extensions.

10.1 Conclusion

In this thesis a mathematical model of the velocity and distance to a target has been investigated and derived from standard kinematics. The necessary simplifications of the model due to limitations of the sensor has been discussed.

The two most popular optical flow algorithms have been presented, with a primary focus on the Lucas-Kanade algorithm due to its popularity. The use of pyramids for calculating optical flow when the motion is large, has been investigated in the case where the Lucas-Kanade algorithm is applied. Furthermore, how the optical flow output can be used in local navigation has been shown. The optical flow can be used to avoid collisions with objects, much like insects do in real life. Another application is the case of tracking a moving object and estimating the angular velocity.

In cases where there is a lack of features to track, optical flow will not be applicable. Hough transform was presented as an alternative to optical flow. Hough transform can be used to detect any parametric curves, but the focus in this thesis was to detect straight lines. The lines can be used to find relative angles between blades or windows, which again can be used in a yaw controller. Furthermore, the use of Hough transform to inspect a building or wind turbine for damages has been discussed.

In addition, an observer capable of estimating the metric velocity and distance to the target has been derived. Through simulations with two different scenarios, the observer successfully estimated the unknown states based on the output of an optical flow algorithm and IMU. The simulation showed that the observer is capable of estimating the unknown states even though the scaled velocity provided by the optical flow algorithm was noisy. Due to lack of optical flow inputs, the observer was not tested on the UAV.

Furthermore, a controller has been derived. The controller receives a desired velocity vector from either the observer or the Hough transform. By using a combination of PID

controllers, one for each output, the controller calculates the desired roll, pitch, yaw and throttle commands. If the wind turbine blade is not in the center of the image, this can be compensated by increasing or decreasing the throttle command value.

The two different applications considered in this thesis, wind turbine blade and building inspection, were simulated. The simulations verified that the controller is able to successfully maneuver the UAV to inspect the blade and building for damage. In the wind turbine blade inspection, the UAV started on the ground and flew to the hub. At the hub, the UAV started inspecting the front of the blade for damages. When it reached the tip of the blade, it flew around and began inspecting the back. In the building inspection simulations, two different search patterns were implemented and tested. The results showed that the parallel and creeping line search produced satisfying results. The UAV was able to inspect the building with adaptive line width.

The different controllers were tested on an UAV platform. The pitch controller kept the UAV at a constant distance with satisfactory result, while the roll controller utilized the camera information to control the UAV to a stationary location. The height controller was tested with both a stationary and time-varying set point. Both tests produced satisfactory results. The in-built controllers did not follow the desired angles perfectly, so the test will serve as a proof of concept. Further fine-tuning of the PID parameters will much likely lead to significantly better results.

10.2 Future work

In this thesis, we have seen that the optical flow algorithm tracking a singular point does not provide any useful information in the case where the UAV moves directly towards or away from the object. Further investigation can be conducted to look at the possibilities of tracking multiple features. By tracking multiple features, the optical flow vectors can be used to sense if the UAV is moving towards or away from the target. The features will expand or contract relative to the center of the image when the UAV moves closer or further away from the object. Thus, the optical flow output can be fused with the accelerometer measurements to provide a more accurate distance estimate.

A pan/tilt system can be installed on the camera without any modification to the code. The only change necessary is to measure the time-varying angles between the UAV and camera, ϕ_c, θ_c, ψ_c. The installation of a pin tilt mechanism makes it possible to remotely control the camera and thus specify where to inspect for damages. The requirement for this set up is a ground station equipped with a computer and a monitor. The computer does not need to be powerful, as all of the necessary computations are done on-board of the UAV. The main objective of the ground station will be to receive the live camera stream and send control signals to the pin/tilt mechanism on the camera.

Recently, 3D Robotics released a new autopilot titled Pixhawk. The Pixhawk features two accelerometers with different sampling frequency. This offers redundancy as well as preventing aliasing due to vibrations. In addition, the Pixhawk features a faster processor. The increased processing power made it possible to implement an EKF. The EKF fusions the readings from the gyroskop, accelerometer, magnetometer, barometer and GPS to

estimate the position, velocity and attitude of the UAV. Consequently, the EKF handles sensor glitches better, as well as more accurate attitude estimation.

Due to lack of provided camera information, the yaw controller has not been tested. However, by manually controlling the yaw, the remaining three controllers can be connected and tested. A natural continuation will be to pan a blade to test the controllers simultaneously. This will also test the ability of the high resolution pictures to detect damages. Furthermore, due to the limited range of the sonar, the estimated height using barometer needs to be investigated further. If the barometer does not provide the necessary accuracy, a LRF has to be considered.

Appendix A

IMU bias and dead reckoning

This appendix will address the source of error in a IMU for position estimates, e.g. bias and drift. Furthermore, the use of dead reckoning as an aiding sensor will be discussed.

A.1 IMU bias

This section addresses the source of error in accelerometer, i.e. noise and bias. The acceleration measurement received from the IMU is not perfect. The measurement received from the accelerometer is given as

$$\mathbf{f}_{measured} = \mathbf{f} + \mathbf{b} + \mathbf{v}$$

where $\mathbf{f}$ is the acceleration of the UAV. Furthermore, $\mathbf{b}$ and $\mathbf{v}$ is the bias and noise of the measurements, respectively. To obtain a position estimate, the acceleration readings can be integrated twice. In theory this will give the correct position. However, since the IMU also measures bias and noise, the integration will lead to position drift. The bias can be estimated using a filter, e.g. Kalman filter, and subtracted from the measurements. Nonetheless, a small amount of bias will always be present and, coupled with noise, will be integrated and lead to drift.

A.2 Dead reckoning

Dead reckoning (DR) is to use inertial sensors to estimate the relative position of an object. DR is self-contained, i.e. not rely on external sensors such as GPS. As shown in the previously section, the IMU signal is not perfect, but will drift due to noise and bias. Hence, inertial navigation is only practical for a few seconds [15]. In order to achieve long term accuracy, additional sensors are required. In global navigation, e.g. air planes, satellites and ships, GPS is the most popular aiding sensor. In cases where the GPS signal is unavailable, other approaches are applicable. Zero velocity update is a technique where the INS is corrected when the velocity is zero, i.e. in a constant position. To determine when the velocity is zero, camera information can be utilized. The optical flow algorithm can detect when the velocity is zero by looking at the displacement of pixels to correct the relative position estimate, see Figure A.1.

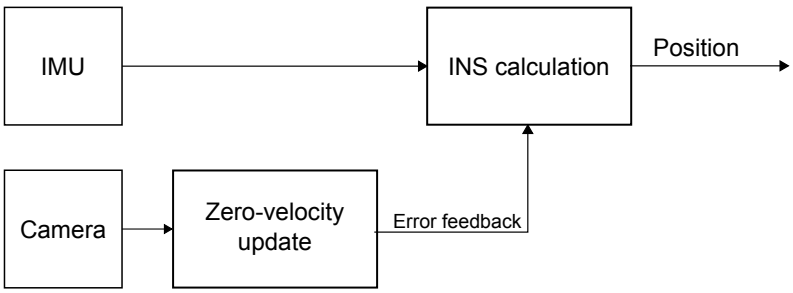


Figure A.1: Block diagram of dead reckoning with zero-velocity update

Figure A.2: IMU embedded in the heel of a firefighter boot. Courtesy of [51]

There are several applications for dead reckoning. The most common is in ship navigation when GPS signal is lost or when cars drives through tunnels and loose GPS signal. Recently, DR has been used in human applications, e.g. fire fighters when they search for survivals in a room filled with smoke. By including an IMU and two force-sensitive resistors in the shoes for zero-velocity update, the INS accomplish long term accuracy [66].

A.2.1 DR as aiding information

Even though optical flow and Hough transform are used when camera information is available, DR can act as an aiding sensor. By fusing the camera information with DR, the UAV is able to fly back to its last position. This is helpful in cases where wind gusts forces the UAV camera out of sight of the wind turbine. Furthermore, a camera is easily affected by sun light. DR can aid the controller in cases where the backlight ruins the visibility of the wind turbine or building.

A.2.2 Around the tip and corners

DR can also be used when the UAV is about to fly around the tip of the blade or around building corners. Neither optical flow nor Hough transform will be available around the tip, thus the UAV can only rely on the estimate velocity and relative position. As previously discussed, the estimate will not be accurate over a period of time. However, the time the UAV spends flying around the tip or corner, is within the accuracy of DR.

Appendix B

Roof lab

The roof lab is a lab on top of A-building at NTNU. The building features a 5×5×3 meter safety net for controlled flying. The net serves to protect both humans and equipment from the UAV. The building is constructed of wood, allowing GPS to communicate with satellites and provides the UAV with navigation data, i.e. position and velocity.

Figure B.1: The UAV lab at the roof

Bibliography

[1] Swarm of Micro Flying Robots. `http://www.sfly.org/`. [Online; accessed 24-November-2013].

[2] Evan D Andersen and Clark N Taylor. Improving mav pose estimation using visual information. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3745–3750. IEEE, 2007.

[3] Jorge Artieda, José M Sebastian, Pascual Campoy, Juan F Correa, Iván F Mondragón, Carol Martínez, and Miguel Olivares. Visual 3-d slam from uavs. *Journal of Intelligent and Robotic Systems*, 55(4-5):299–321, 2009.

[4] John L Barron, David J Fleet, and Steven S Beauchemin. Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77, 1994.

[5] Alison Brown, Ben Mathews, and Dien Nguyen. Integrated gps/ins/star tracker space navigation system using a software defined radio.

[6] Robert Grover Brown and Patrick YC Hwang. *Introduction to random signals and applied Kalman filtering*. John Wiley & Sons, Inc, 2012.

[7] Pedro Castillo, Rogelio Lozano, and Alejandro Dzul. Stabilization of a mini rotorcraft with four rotors. *IEEE Control Systems Magazine*, 25(6):45–55, 2005.

[8] Lance Champagne, R Greg Carl, and Raymond Hill. Agent models ii: search theory, agent-based simulation, and u-boats in the bay of biscay. In *Proceedings of the 35th conference on Winter simulation: driving innovation*, pages 991–998. Winter Simulation Conference, 2003.

[9] Haiyang Chao, Yu Gu, and Marcello Napolitano. A survey of optical flow techniques for uav navigation applications. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 710–716. IEEE, 2013.

[10] Metrology Resource Co. Metrology Sensor Products - MRL3. `http://www.metrologyresource.com/laser-sensor-MRL3.php`. [Online; accessed 02-March-2014].

[11] Peter Corke. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer, 2011.

[12] Vladimir N Dobrokhodov, Isaac I Kaminer, Kevin D Jones, and Reza Ghabcheloo. Vision-based tracking and motion estimation for moving targets using unmanned air vehicles. *Journal of guidance, control, and dynamics*, 31(4):907–917, 2008.

[13] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[14] Olav Egeland and Jan Tommy Gravdahl. *Modeling and simulation for automatic control*, volume 76. Marine Cybernetics Trondheim, Norway, 2002.

[15] Naser El-Sheimy and Xiaoji Niu. The promise of mems to the navigation community. *Inside GNSS*, 2(2):46–56, 2007.

[16] WSP Fernando, Lanka Udawatta, and Pubudu Pathirana. Identification of moving obstacles with pyramidal lucas kanade optical flow and k means clustering. In *Information and Automation for Sustainability, 2007. ICIAFS 2007. Third International Conference on*, pages 111–117. IEEE, 2007.

[17] Gerardo Flores, Shuting Zhou, Rogelio Lozano, and Pedro Castillo. A vision and gps-based real-time trajectory planning for mav in unknown urban environments. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 1150–1155. IEEE, 2013.

[18] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011. ISBN 978-1-119-99149-6.

[19] Ben Galvin, Brendan McCane, Kevin Novins, David Mason, and Steven Mills. Recovering motion fields: An evaluation of eight optical flow algorithms. In *BMVC*, volume 98, pages 195–204, 1998.

[20] Harris Gasparakis. OpenCV-CL: Computer vision with OpenCL acceleration. `http://developer.amd.com/community/blog/2013/07/09/opencv-cl-computer-vision-with-opencl-acceleration/`, 2013. [Online; accessed 13-December-2013].

[21] Paolo Robuffo Giordano, Alessandro De Luca, and Giuseppe Oriolo. 3d structure identification from image moments. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 93–100. IEEE, 2008.

[22] Audrey Giremus, Arnaud Doucet, Vincent Calmettes, and J-Y Tourneret. A rao-blackwellized particle filter for ins/gps integration. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 3, pages iii–964. IEEE, 2004.

[23] Volker Grabe, Heinrich H Bülthoff, Paulo Robuffo Giordano, et al. A comparison of scale estimation schemes for a quadrotor uav based on optical flow and imu measurements. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'2013*, 2013.

[24] Zhihai He, Ram Venkataraman Iyer, and Phillip R Chandler. Vision-based uav flight control and obstacle avoidance. In *American Control Conference, 2006*, pages 5–pp. IEEE, 2006.

[25] Bruno Herisse, F-X Russotto, Tarek Hamel, and Robert Mahony. Hovering flight and vertical landing control of a vtol unmanned aerial vehicle using optical flow. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 801–806. IEEE, 2008.

[26] Dominik Honegger, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *Intl. Conf. Robotics and Automation*, 2013.

[27] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1):185–203, 1981.

[28] Paul VC Hough. Method and means for recognizing complex patterns, December 18 1962. US Patent 3,069,654.

[29] Stefan Hrabar, Gaurav S Sukhatme, Peter Corke, Kane Usher, and Jonathan Roberts. Combined optic-flow and stereo-based navigation of urban canyons for a uav. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3309–3316. IEEE, 2005.

[30] CorDex Instruments. Why use a Laser Distance Meter? - Understanding the technology. `http://www.transcat.com/technical-reference/PDFs/Application%20Notes/cordex-laser-distance-meters.pdf`. [Online; accessed 01-March-2014].

[31] Dewi Jones. Power line inspection-a uav concept. In *Autonomous Systems, 2005. The IEE Forum on (Ref. No. 2005/11271)*, pages 8–pp. IET, 2005.

[32] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[33] Jonathan Kelly and Gaurav S Sukhatme. Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration. *The International Journal of Robotics Research*, 30(1):56–79, 2011.

[34] Jonghyuk Kim and Salah Sukkarieh. Real-time implementation of airborne inertial-slam. *Robotics and Autonomous Systems*, 55(1):62–71, 2007.

[35] AD King. Inertial navigation-forty years of evolution. *GEC review*, 13(3):140–149, 1998.

[36] M Koifman and IY Bar-Itzhack. Inertial navigation system aided by aircraft dynamics. *Control Systems Technology, IEEE Transactions on*, 7(4):487–493, 1999.

[37] Sven Lange, Niko Sunderhauf, and Peter Protzel. A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6. IEEE, 2009.

[38] Kyuho Lee. *Development of Unmanned Aerial Vehicle (UAV) for Wildlife Surveillance*. PhD thesis, University of Florida, 2004.

[39] Institut fÃijr Informationsverarbeitung Leibniz UniversitÃđt Hannover. System for early damage and ice detection for rotor blades of offshore wind turbines. http://www.tnt.uni-hannover.de/project/DamDet/. [Online; accessed 29-November-2013].

[40] Frederik Stendahl Leira. Infrared object detection & tracking in uavs. 2013.

[41] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[42] Daniel Magree, John G Mooney, and Eric N Johnson. Monocular visual mapping for obstacle avoidance on uavs. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 471–479. IEEE, 2013.

[43] MathWorks. vision.PointTracker System object. http://www.mathworks.se/help/vision/ref/vision.pointtrackerclass.html. [Online; accessed 28-November-2013].

[44] MaxBotix. XL-MaxSonar®- EZ^{TM}Series . http://maxbotix.com/documents/XL-MaxSonar-EZ_Datasheet.pdf. [Online; accessed 26-February-2014].

[45] Leonard A McGee and Stanley F Schmidt. Discovery of the kalman filter as a practical tool for aerospace and industry. *NASA Technical Memorandum 86847*, 1985.

[46] DA Mercado, G Flores, P Castillo, J Escareno, and R Lozano. Gps/ins/optic flow data fusion for position and velocity estimation. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 486–491. IEEE, 2013.

[47] Torsten Merz, Simone Duranti, and Gianpaolo Conte. Autonomous landing of an unmanned helicopter based on vision and inertial sensing. In *Experimental Robotics IX*, pages 343–352. Springer, 2006.

[48] Faraz M Mirzaei and Stergios I Roumeliotis. A kalman filter-based algorithm for imu-camera calibration: Observability analysis and performance evaluation. *Robotics, IEEE Transactions on*, 24(5):1143–1156, 2008.

[49] Iván Fernando Mondragón, Pascual Campoy, Carol Martinez, and Miguel A Olivares-Méndez. 3d pose estimation based on planar object tracking for uavs control. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 35–41. IEEE, 2010.

[50] Gabriel Nützi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Fusion of imu and vision for absolute scale estimation in monocular slam. *Journal of intelligent & robotic systems*, 61(1-4):287–299, 2011.

[51] The University of Michigan. Section 4: Pedestrian tracking. URL http://mrl.engin.umich.edu/PE_Indoor.html.

[52] LightWare Optoelectronics. SF02 Laser rangfinder Product manual. http://www.lightware.co.za/download/doc/SF02%20-%20Laser%20Rangefinder%20Manual%20Rev%2001.pdf. [Online; accessed 05-March-2014].

[53] Renato S Ornedo, Kenneth A Farnsworth, and Gurpartap S Sandhoo. Gps and radar aided inertial navigation system for missile system applications. In *Position Location and Navigation Symposium, IEEE 1998*, pages 614–621. IEEE, 1998.

[54] PandaBoard. Platform. `http://pandaboard.org/node/300/#PandaES`. [Online; accessed 13-December-2013].

[55] NP Papanikolopoulos, Bradley Nelson, and PK Khosla. Monocular 3-d visual tracking of a moving target by an eye-in-hand robotic system. In *Decision and Control, 1992., Proceedings of the 31st IEEE Conference on*, pages 3805–3810. IEEE, 1992.

[56] Richard H Parvin. *Inertial navigation*, volume 9. Van Nostrand, 1962.

[57] Dr. RÃijdiger Paschotta. Group Delay. `http://www.rp-photonics.com/group_delay.html`. [Online; accessed 05-March-2014].

[58] José Pinto, Pedro Calado, José Braga, Paulo Dias, Ricardo Martins, Eduardo Marques, and JB Sousa. Implementation of a control architecture for networked vehicle systems. In *Navigation, Guidance and Control of Underwater Vehicles*, volume 3, pages 100–105, 2012.

[59] José Pinto, Joao Sousa, Frédéric Py, and Kanna Rajan. Experiments with deliberative planning on autonomous underwater vehicles. In *IROS 2012 Workshop on Robotics for Environmental Monitoring, Vilamoura, Portugal*, 2012.

[60] Paul Pounds, Robert Mahony, and Peter Corke. Modelling and control of a quadrotor robot. In *Proceedings Australasian Conference on Robotics and Automation 2006*. Australian Robotics and Automation Association Inc., 2006.

[61] Sven Rönnbäck. Developement of a ins/gps navigation loop for an uav. *Master's thesis*, 81, 2000.

[62] OS Salychev, VV Voronov, ME Cannon, R Nayak, and G Lachapelle. Low cost ins/gps integration: Concepts and testing. In *Proceeding of the Institute of Navigation National Technical Meeting*, pages 98–105, 2000.

[63] Srikanth Saripalli and Gaurav S Sukhatme. Landing a helicopter on a moving target. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2030–2035. IEEE, 2007.

[64] Dr. Mubarak Shah. Lecture 07 - Pyramids. `http://www.youtube.com/watch?v=NiGcuurpV5o`, 2012. [Online; accessed 13-November-2013].

[65] Dr. Mubarak Shah. Lecture 06 - Optical Flow. `http://www.youtube.com/watch?v=5VyLAH8BhF8`, 2012. [Online; accessed 13-November-2013].

[66] Isaac Skog, Peter Handel, J-O Nilsson, and Jouni Rantakokko. Zero-velocity detectionâĂŤan algorithm evaluation. *Biomedical Engineering, IEEE Transactions on*, 57 (11):2657–2666, 2010.

[67] Mandyam V Srinivasan, Javaan S Chahl, Keven Weber, Svetha Venkatesh, Martin G Nagle, and Shao-Wu Zhang. Robot navigation inspired by principles of insect vision. *Robotics and Autonomous Systems*, 26(2):203–216, 1999.

[68] Andrew M Uhl. Project 3, Topic D: Seafloor Mapping with Sonar. `http://www.personal.psu.edu/amu5000/geog160-project3.htm`, 2007. [Online; accessed 26-February-2014].

[69] Shimon Ullman. *The interpretation of visual motion.* Massachusetts Inst of Technology Pr, 1979.

[70] Binhai Wang, Xiguang Chen, Qian Wang, Liang Liu, Hailong Zhang, and Bingqiang Li. Power line inspection with a flying robot. In *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, pages 1–6. IEEE, 2010.

[71] Yoko Watanabe, Patrick Fabiani, and Guy Le Besnerais. Simultaneous visual target tracking and navigation in a gps-denied environment. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6. IEEE, 2009.

[72] Stephan Weiss, Markus W Achtelik, Simon Lynen, Margarita Chli, and Roland Siegwart. Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 957–964. IEEE, 2012.

[73] Eric W. Weisstein. Hessian normal form, . URL `http://mathworld.wolfram.com/HessianNormalForm.html`.

[74] Eric W. Weisstein. Line-line intersection, . URL `http://mathworld.wolfram.com/Line-LineIntersection.html`.

[75] Ick Ho Whang and Won-Sang Ra. Simple altitude estimator using air-data and gps measurements. *Proceedings of the 17th World Congress of the International Federation of Automatic Control*, 17(1):4060 – 4065, 2008.

[76] Robert Wolf, Guenter W Hein, Bernd Eissfeller, and Erwin Loehnert. An integrated low cost gps/ins attitude determination and position location system. In *Proceedings of the 9th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 1996)*, pages 975–981, 1996.

[77] Jiangjian Xiao, Changjiang Yang, Feng Han, and Hui Cheng. Vehicle and person tracking in uav videos. In *Classication of Events, Activities, and Relationships Evaluation and Workshop*, 2007.

[78] Takeshi Yamasaki, Hiroyuki Takano, and Yoriaki Baba. Robust path-following for uav using pure pursuit guidance. *Aerial Vehicles*, pages 671–690.

[79] Shaowu Yang, Sebastian A Scherer, Konstantin Schauwecker, and Andreas Zell. Onboard monocular vision for landing of an mav on a landing site specified by a single reference image.

[80] Simon Zingg, Davide Scaramuzza, Stephan Weiss, and Roland Siegwart. Mav navigation through indoor corridors using optical flow. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3361–3368. IEEE, 2010.