NTNU – Trondheim
Norwegian University of
Science and Technology

A Computer Vision Approach for Autonomous Wind Turbine Inspection using a Multicopter

Martin Stokkeland

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

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

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

MSC THESIS DESCRIPTION SHEET

Name:	Martin Stokkeland
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Bruk av bildebehandling for autonom inspeksjon av vindmøller ved hjelp av et multicopter
Thesis title (English):	A computer vision approach for autonomous wind turbine inspection using a multicopter

Thesis Description: Investigate and study methods for successful and efficient image processing techniques to be used in local navigation from a UAV-platform like a multicopter (quad/hexa). In particular, consider the mission of autonomous wind-turbine inspection. *Emphasis on object tracking and recognition as a solution basis.*

The following items should be considered:

1. Discuss key features of the wind-turbine to be used as features for image processing. Define characteristics of a turbine hub.
2. Overall system description with detailed module interaction schemes and protocols.
3. Literature review on current state-of-the-art image processing techniques for this application.
4. Design and implement **object tracking**-based algorithms for local navigation in OpenCV on a desktop computer, easy portability between platforms are a requirement. Preferably, details concerning video drivers and devices on a Linux-based operating system.
5. The implementation should be done in the operating-framework Dune.
6. Discuss how the output from such an algorithm can be used to create mathematical observers (brief).
7. Investigate and discuss necessary hardware (cameras, video converters, etc)
8. The results should be verified by experiments.
9. Conclude findings in a report. Include Matlab/C-code as digital appendices together with a user-guide.

Start date:	2014-01-21
Due date:	2014-06-17

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 studies the mission of autonomous inspection of a wind turbine using a multicopter. Emphasis was placed on recognition and tracking using image processing methods. The Hough line transform was used to extract features of the wind turbine. Hub position was estimated by an algorithm tailored to identify the three-point star resemblance and was tracked by utilizing the Kalman filter. Distance and yaw orientation of the wind turbine were estimated using the pinhole camera model and coordinate transformations. Restricting computational demand was a goal in the program design. Experiments showed accurate position tracking at long range, but with deteriorating performance as range was decreased. Lack of distinctive measurable lengths in the image caused inaccuracy in estimation of distance and yaw orientation. Execution frequency of below 7 Hz was achieved on a single-board computer which was found to be sufficient for reliable control in flight.

Sammendrag

Denne oppgaven utforsker utføring av autonom inspeksjon av vindmøller ved hjelp av et multikopter. Gjenkjenning og tracking ved hjelp av bildebearbeiding var i hovedfokus. Hough line transform ble brukt til å registrere vindmøllens trekk. Navets posisjon ble estimert av en algoritme designet for å gjenkjenne vindmøllens stjerneform og ble tracket ved hjelp av et Kalman filter. Avstand og yaw orientering av vindmøllen ble estimert ved hjelp av pinhole camera model og koordinattransformasjoner. Programmet ble designet med kjøretid tatt i betraktning. Eksperimenter viste nøyaktig estimering av posisjon under lang avstand, men med forringet ytelse ved minkende avstand. Mangel på lett gjenkjennelige målbare avstander i bildet førte til unøyaktig estimering av avstand og yaw vinkel. Det ble oppnådd kjørefrekvens på under 7 Hz på en single-board computer, hvilket viste seg å være tilstrekkelig for pålitelig kontroll under flyvning.

Preface

This thesis concludes the final step in my journey towards 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 further like to thank my co-supervisor, Ph.D. student Kristian Klaussen, for sound advice and valuable feedback, and engineer Lars Semb for his aid in gathering video data.

I would also like to thank my family for their support throughout my years of study.

Contents

List of Figures

List of Tables

List of Acronyms

UAV Unmanned Aerial Vehicle

MAV Micro Air Vehicle

IMU Inertial Measurement Unit

HT Hough Transform

PHT Probabilistic Hough Transform

PPHT Progressive Probabilistic Hough Transform

SBC Single-Board Computer

GPS Global Positioning System

APM ArduPilot Mega

USB Universal Serial Bus

Chapter 1

Introduction

1.1 Background and Motivation

Over the recent years the unmanned aerial vehicle (UAV) has emerged as a subject of great interest in robot research. The multicopter in particular, possesses a multitude of properties which makes it attractive for a wide range of operations. It has excellent maneuverability thanks to the distributed rotor structure. This combined with the light weight and small size makes it suitable for many tasks which would otherwise not be possible.

Among fields where multicopters have been employed are military operations, search and rescue, surveillance, structure inspection and even pizza delivery. In this work, however, the focus will be on wind turbine inspection, a form of structure inspection. For a similar case, inspection of power lines, Williams et al. (2001) described several reasons why using a small UAV is preferable. One reason is the hazards involved in flying a manned helicopter at close range. An autonomous or remotely controlled multicopter on the other hand would be able to approach the target closely without high risk. In case of impact the damage would be minor. Another argument is the cost, which is significantly smaller than it would be by using a heavy manned helicopter or a time consuming manual inspection by foot.

The presently utilized approaches of wind turbine inspection are mainly examining through telescopes, by climbing or by using remotely controlled unmanned aerial vehicles (UAVs). In order to eliminate the need of human time investment it would be preferable to perform this task autonomously. For such a task, the UAV is commonly provided camera sensors, which accompanied with an Inertial Measuring Unit (IMU) and the Global Positioning System (GPS) provide powerful means for solving navigation and guidance problems. This paper will focus on how a computer vision system can be

implemented in order to effectively track a wind turbine, and produce results which can be provided as useful input to the controller.

1.2 Previous Work

The area of onboard computer vision has grown tremendously in recent past as computers have progressed to the point of handling such complex tasks at a decent rate. Liu and Dai (2010) mentions several computer vision applications for UAV's including visual servoing, optical flow, visual navigation, target detection and tracking, and simultaneous localization and mapping (SLAM).

In many cases the computer vision methods are considered with regard to a GPS denied environment. Caballero et al. (2009) approached this issue by identifying natural landmarks through feature tracking, which were integrated into a SLAM scheme. By using such a scheme one can divert from the use of global states and instead define states relative to a node in the map, as explained by Leishman et al. (2013).

Another topic of interest is how computer vision can be used to complement the data provided by the IMU. Optical flow algorithms are a common approach in this regard, where for instance Kendoul et al. (2009) estimated velocity and position using a downward facing camera. By employing stereo vision more sophisticated algorithm involving depth information can be designed, as presented by Schauwecker and Zell (2013) whose algorithm also provided pose estimation.

Apart from navigation and guidance, the visual information can provide knowledge which is not easily obtained in any other way. Magree et al. (2013) illustrates an example of how computer vision provides data for obstacle avoidance. Another issue of interest has been finding safe landing positions for emergency landing or uncharted environments, where Mejias and FitzGerald (2013) has proposed a segmentation based solution.

1.3 Contribution and Scope of this Report

To perform the inspection mission for the hexacopter system presented in this work, it is of interest to describe the position and orientation relative to a wind turbine during flight. The contribution lies in elaborating on image processing methods for which the wind turbine can be recognized, and how its features can provide position and orientation estimations. Videos of a wind turbine were recorded at Bessakerfjellet wind farm (appendix B.2) for

testing and validation. Tools available in the OpenCV library (section 5.2.2) will utilized in forming the solution. Emphasis will be placed on keeping computational demand to a level which can be handled by the carried single-board computer (SBC) which is mounted on the hexacopter.

The scope of this work is limited to what can be called the first part of an inspection. That is, to approach from an initial point where the UAV has arrived in adjacency of the wind turbine using (for instance) GPS navigation, to the destination point which is directly in front of the hub of the wind turbine. The reason for this is that the project originally split in two, where the objective of the other student was to continue after the first part to study how to follow and inspect each of the individual blades. However, the other student resigned from the project, but a brief discussion will also be given on blade inspection.

1.4 Organization of this Report

First, the characteristics of wind turbine are be investigated to get a notion of what information is available to base algorithms upon. Next, some concepts and approaches from the field of computer vision are introduced. The introductions are more thorough for concepts of higher relevance. In chapter 5 a description of the complete system is given. Some methods for recognition and tracking are discussed in chapter 6. Based on the conclusions an algorithm is designed and explained. Chapter 7 suggest a method for approaching the wind turbine in regard of control commands. Additionally, methods for distance and yaw angle estimation are discussed. In chapter 8, the final program design is described, where algorithms discussed in the previous analyses are integrated. Results from experiments are presented in chapter 9. Position, distance and yaw angle estimations obtained from video input are presented, along with execution time analysis and flight control performance. At the end follows final discussion and conclusion.

Chapter 2

Notation and Coordinate Frame Definitions

2.1 Notation

- Vectors and matrices are written in bold, e.g. $\mathbf{A}$, $\mathbf{v}$. All vectors are column vectors. The transpose of a vector $\mathbf{v}$ is denoted $\mathbf{v}^T$.
- Coordinate frames are given in italic type, and superscript specifies the coordinate frame for the given vector. E.g $\mathbf{v}^n$ specifies the vector $\mathbf{v}$ given in coordinate frame n.
- Subscripts are used on various occasions. For a sequence of vectors the subscript k, in $\mathbf{x}_k$, determines the position of the vector in the sequence. Subscripts are also used to extract single elements from a vector, as in v_n which denotes the value of vector $\mathbf{v}$ along the n-axis. Otherwise the meaning of the subscript is specified.
- Rotations between coordinate frames are performed using rotation matrices, e.g. a rotation matrix denoted $\mathbf{R}_b^a$ transforms a vector from b to a according to

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

  For a single axis rotation the rotation matrix may be denoted as $\mathbf{R}_b^a(\alpha)$, where α specifies an angle of rotation around a defined axis.
- Axis lines are denoted with capital letters to differentiate from coordinate values, e.g. for a coordinate system in n with axis lines X^n, Y^n, Z^n a point is expressed using coordinates $(x, y, z)^n$.

2.2 Coordinate Frame Definitions

This section defines the various coordinate frames which will be used throughout this report. They are illustrated in figure 2.1.

2.2.1 Body frame, *b*

The body frame follows the standard roll-pitch-yaw convention for vehicles. It is centered at the geometrical center of the UAV. The x-axis points longitudinally from back to front, the y-axis points laterally from left to right and the z-axis points from top to bottom.

2.2.2 Camera frame, *c*

The camera will be mounted on the UAV facing forwards. Therefore, to avoid confusion, the camera frame is chosen so that the axes are parallel to those of the body frame. It will, however, be centered at the camera itself. Since the camera will be mounted close to the center of the UAV, the distinction between the two frames is usually negligible at the relevant scale.

2.2.3 Image frame, *i*

The image frame is the 2D image projected by the camera. It is centered in the center of the image. The y-axis and z-axis are aligned parallel to the y-axis and z-axis of the camera frame. The actual image produced by the camera is a discretized and rescaled segment of this plane.

2.2.4 Wind turbine frame, *w*

This frame is centered at the hub center of the wind turbine. The z-axis points from top to down, similarly to the body frame. However, the x-axis points *from front to back*. It is chosen this way so that X^w and X^b align when the UAV is directly in front of and facing the hub center. The y-axis lies laterally from the right side to the left side of the wind turbine (which is left to right from the UAV's point of view when facing the front).

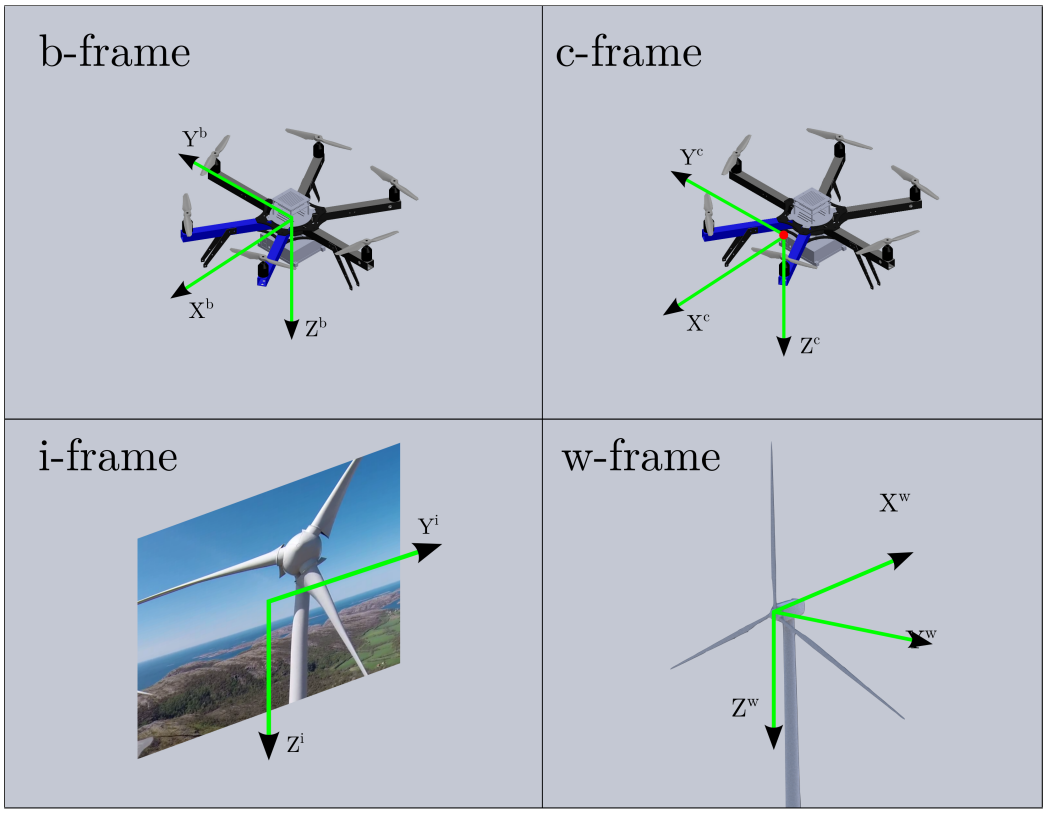


Figure 2.1: Illustrations of the coordinate frames. The red dot depicts the mounting point of the camera and defines the origin of the c-frame. (Hexacopter modelled by Kristian Klaussen, Wind Turbine modelled by Torjus Sveier Ottemo.

Chapter 3

Key Features of the Wind Turbine

In order to achieve successful recognition and tracking it is of key importance to identify the characteristics which distinguish the object from its surroundings. The following properties of the wind turbine will now be investigated:

- Texture
- Color
- Geometry
- Size
- Movement
- Point of view
- Environment

Texture

The smooth surface of a wind turbine has virtually no texture. When at close range e.g. following when following a blade, this can cause problems for camera based motion detection such as optical flow algorithms. In this report however, the main focus will be tracking at a longer distance where this is no significant issue. It should on the contrary provide a benefit, as the wind turbine is more distinctively separated from the background.

Color

Despite being white in color, a wind turbine can appear in any shade of gray depending on the lighting conditions. However, no particular color tone should be prominent if a proper camera is used.

Geometry

The distinct shape of the wind turbine is of great importance as it possesses many features which can be applied to image processing algorithms. In particular, the straight lines along the rotor blades and the tower are favourable subjects for edge or line detection algorithms. Furthermore, the radial shape composed by the tower and the blades provides solid foundation for many approaches. In addition, a similar shape is not expected to appear elsewhere in the environment (except for other wind turbines which might be present).

Size

Since the GPS system is more practical to use at a broad scale, the UAV is expected to be close to the wind turbine when the algorithm is used. At this range the wind turbine, when in view, should cover a large portion of the image frame and stretch out of the frame when at close distances. Consequently, as long as the UAV is not moving too fast and is facing the wind turbine, it should be clearly visible in the image. The size of which the wind turbine appears in the image can also serve as an estimate for the distance from the UAV.

Movement and time variance

In a wind turbine inspection scenario it is assumed that the wind turbine is stopped for maintenance so that the blades are not rotating. On the other hand, when the UAV moves it is expected to observe a movement of the wind turbine in relation to the background. However, since there initially is significant distance between the UAV and the wind turbine it would require the UAV to move a significant distance before this effect is observed. In addition, a textured background would be required in order to detect this movement, which would make an algorithm based on this principle dependent on the environment.

Point of view

It will generally be assumed that the UAV is positioned in front of the wind turbine such that the rotor blades or tower do not overlap. When directly in the front or back the angle between the blades are 120 degrees. A change of perspective skews the image altering both angles and proportions, thus providing cues for estimating the yaw orientation of the wind turbine.

Environment

Under clear weather conditions a blue sky serves as excellent contrast to the wind turbine, whereas clouds and gloomy weather might interfere with detection. Depending on the height and the pitch angle of the camera, the horizon may appear at varying height in the image. If the horizon appears too close to the hub, it is expected to interfere with blade detection. Generally it is preferred that the least possible amount of texture is present in the background and that the color does not overlap with the color of the wind turbine.

Chapter 4

Theory

In this chapter the methods deemed relevant for the project are presented. Emphasis is placed on their general functionality and their anticipated benefits. They will later be brought up at relevant points to discuss to what degree they are applicable for the task in question.

4.1 Hough Line Transform

4.1.1 Standard Hough line transform

The Hough transform (HT) is a widely utilized technique for feature detection. It is based on a statistic principle in which a feature is identified by accumulating a specific amount of votes from points in the image which contribute to this feature.

A single channel binary image is most commonly used as input for the method. Therefore it is usually required to perform an edge detection algorithm on the original image before attempting the Hough transform, e.g. the Canny edge transform (section 4.3.1).

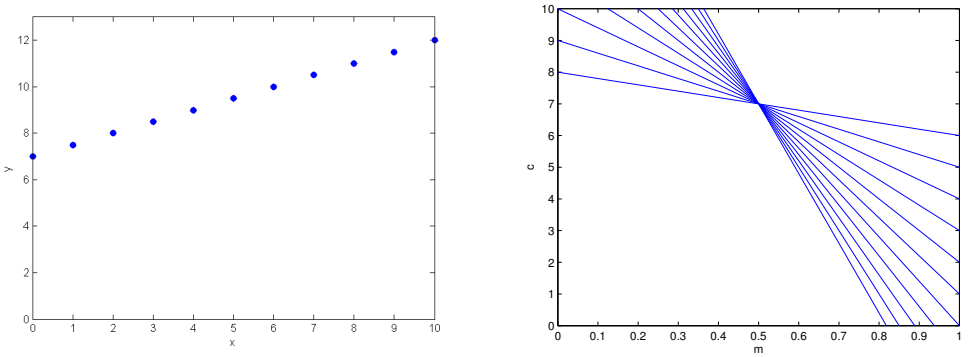
The standard Hough transform works by utilizing a parametrized function for the feature in question. A line can for instance be represented by equation (4.1), which is the form of the original Hough transform patent (Hough, 1959).

$$c(m) = y - m \cdot x \tag{4.1}$$

The parameters m and c denote slope and intersection respectively. Other features which can be parameterized in a similar manner can also be detected by altering equation (4.1) accordingly. For the purpose of wind turbine tracking the focus is limited to line and circle detection.

Using the chosen parametrization, the program next works through all active pixels of the image. An active pixel refers to one of the edge pixels which was produced beforehand by the chosen edge detection algorithm. For each active pixel, the set of all lines which pass through the pixel is found. This is done by iterating through the range of one parameter (m) and calculating the corresponding second parameter (c). Essentially, this leads to a one-to-many mapping from the image space to the parameter space. In this case the result is that every pixel in the image space maps to a line in the mc-space, where each line represents the set of all possible lines which can pass through the pixel. This is seen in figure 4.1 where every pixel in the image space (a) produces a corresponding set in the mc-space (b).

An intersection of these sets would mean that a common line exists for all pixels whose line sets participate in the intersection. Such is the case in figure 4.1b, where the point of intersection parameterizes the one line which connects the pixels in figure 4.1a.



(a) (b)

Figure 4.1: (a): The image space, containing some points arranged in a line. (b): The parameter (m,c) space, containing all possible lines which pass through the points from the image space. The intersection indicates the one line which is common for all points, i.e. the line formed by the points.Illingworth and Kittler (1988)

In practice, this is implemented as a voting procedure, referred to as the voting stage of the algorithm. Every pixel votes for the lines which can possibly pass through it. These votes are passed to an accumulator which collects the results from all pixels. Pseudocode for the voting procedure is in algorithm 1.

After running the voting procedure the accumulator is filled with votes for each parameter combination as shown in table 4.1. The result is merely a

Algorithm 1 The voting stage of the standard Hough transform

```
for all pixels in image do
    if pixel is active then
        for entire range of parameter m do
            c = pixel.y − m · pixel.x
            accumulator(m, c) ← accumulator(m, c) + 1
```

Table 4.1: The accumulated votes for each line candidate.

10	1	1	2	2	1	0	0	0	0	0	0
9	1	1	2	3	5	0	0	0	0	0	0
8	1	1	1	2	5	0	0	0	0	0	0
7	0	0	0	0	0	11	4	2	1	1	0
6	0	0	0	0	0	0	5	2	2	1	1
5	0	0	0	0	0	0	2	3	1	1	1
4	0	0	0	0	0	0	0	2	2	1	1
3	0	0	0	0	0	0	0	2	2	2	1
2	0	0	0	0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	0	0	2	1	1
	0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**

discrete version of the continuous representation shown in 4.1b. In this case, the line of interest is clearly identified by the high number of votes. Generally, the lines are found by extracting the local maxima from the accumulator. A common way to achieve this is through thresholding. This works simply by setting a threshold, where all lines whose votes surpass this threshold value are accepted. It is commonly referred to as the Hough threshold and is usually one of the input parameters for the Hough transform algorithm.

The ideal value for the Hough threshold is highly related to the properties of the image in question. In particular, long lines receive many votes compared to the rest of the image, which will yield good results for a wider range of Hough threshold selections. On the other hand, an image containing many features in addition to the lines of interest, will narrow the range of threshold values that provide good results. Some versions of the Hough transform use a relative threshold approach which instead utilizes local maxima in the accumulator. In the wind turbine inspection scenario however, it is beforehand known that the lines formed by the turbine appear at significant length in the image, and few competing lines are assumed to be formed by the background. This consistency indicates a possibility to experimentally tune the threshold to a value which performs well in a general case.

An issue which needs to be addressed in this approach is the non-linearity of the line parametrization, (4.1). It is apparent that a line approaches a vertical angle, the m-parameter blows up. This is solved by using polar

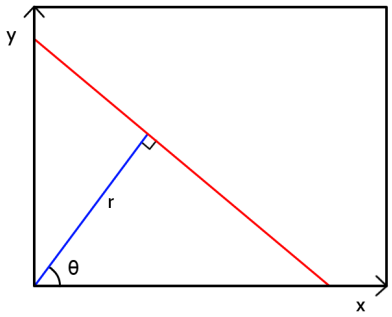


Figure 4.2: Polar coordinate representation of a line.

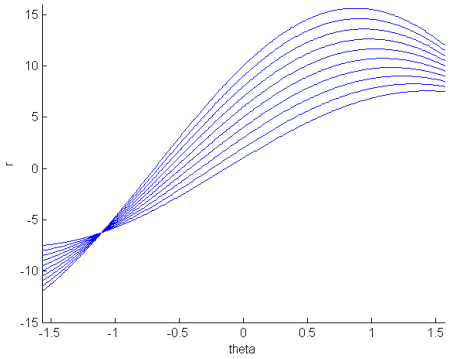


Figure 4.3: The accumulated votes for each line candidate using polar parametrization.

coordinates instead, resulting in the new parametrization shown in (4.2). Duda and Hart (1972) presented this form, and has since become the general approach for the Hough line transform.

$$r(\theta) = x \cdot cos\theta + y \cdot sin\theta \tag{4.2}$$

With the polar parametrization each point now maps to a sinusoidal wave in the parameter space, rather than a straight line as was seen before. The most probable line candidate still appears as the point of intersection in the parameter space, similarly as before. The difference is that the result must be interpreted as a polar representation of a line (figure 4.2).

4.1.2 Probabilistic Hough line lransform

The Standard Hough line transform described in the previous section forms the basic idea of the method. In order to improve efficiency while maintaining performance, numerous variations of the Hough transform (HT) have been studied including but not limited to probabilistic HT, adaptive HT, randomized HT along with numerous combinations and alterations of these. Many of these methods are designed to perform well under high dimension features, however the focus in this report will be on the method available in the OpenCV library as *HoughLinesP()* which efficiently detects lines (2D). This method builds further upon the probabilistic Hough transform.

Originally, the probabilistic Hough transform was proposed by (Kiryati et al., 1991). The idea is that instead of using all the N points to account for the votes, only a randomly selected subset n (where n $<$ N) is used. By examining the operations required for the algorithm the benefits are explained. The method runs for $O(N \cdot M_\theta)$ operations in the voting stage and $O(M_\rho \cdot M_\theta)$ in the search stage (finding the highest votes), where M_ρ and M_θ is the resolution of ρ and θ respectively in the accumulator. Since the voting stage usually is dominant, a nearly linear reduction can be obtained when reducing N to the subset n.

The study by (Kiryati et al., 1991) further showed that this approach exhibits a threshold effect where the performance abruptly jumps from poor to great for a certain value of n. This value is highly problem dependent, but was in their experiments often found at an n to N ratio below 0.1. In other words, it was shown that the probabilistic method has the potential to drop the computation time below 1/10 without yielding any significant performance drop. Thus, by making an educated guess or by examining the problem properties beforehand, one can select a suitable subset ratio for the probabilistic Hough transform. It should however be mentioned that in the study correct outputs were strictly predefined in the experiments, while in a general image example it is somewhat vague and subjective what should actually be classified as a line.

4.1.3 Progressive probabilistic Hough line transform

The progressive probabilistic Hough transform (Matas et al., 2000), builds further upon the probabilistic Hough transform. In OpenCV an implementation of the PPHT is available as *HoughLinesP*, which will be used in the presented program.

An important distinction from the PHT is that the PPHT does not require a subset ratio (see section 4.1.2) to be selected beforehand. Rather than randomly extracting the subset n immediately, the PPHT exploits the knowledge that there exists a threshold effect for the n to N ratio. So instead the algorithm selects points randomly until end conditions are satisfied, where the n to N ratio hopefully ends up close to the threshold effect area.

Achieving a low n to N ratio means that a minimal amount of votes are cast. In order to reduce the amount of votes the PPHT utilizes another procedure. When a line is detected, the algorithm searches for other points which further contribute to the same line. All the points which are found are then denied voting rights, thus reducing the subset n. By this approach, the worst case scenario would be if no lines are detected i.e. all points are presumably caused by noise, in which case all points would cast votes such that n = N. On the other hand, having many lines present will gradually decrease the subset n for each line which is detected and thus resulting in a low n to N ratio.

PPHT is available in OpenCV as the function *HoughLinesP()*. It transforms detected lines back from polar to Cartesian coordinates and also keeps track of the outermost points of each line. In this way start and end points of each line is readily available which is particularly useful when examining how the lines are organized in relation to each other.

4.2 Hough Circle Transform

The Hough circle transform is in many ways analogous to the Hough line transform. When describing the Hough line transform, it was mentioned that the parametrized function can be altered to detect other shapes instead of lines. Thus the circle transform can be derived by utilizing the circle parametrization:

$$(x-a)^2 + (y-b)^2 = r^2 \tag{4.3}$$

One observes that in contrast to the line parametrization, a circle requires three parameters instead of two; radius (r), center x-position (a) and center y-position (b). This is an issue as the accumulator becomes a three dimensional volume, which would lead to considerable computational demand using the same approach as for the line transform.

In order to handle the dimensionality problem, the OpenCV implementation has utilized the *Hough gradient method* (Kimme et al., 1975). This approach makes clever use of the image gradients to boost efficiency. A brief explanation follows.

As usually done with the normal Hough transform, the method begins by obtaining the binary edge map using the Canny edge detector (4.3). Additionally, gradients are computed by using first order Sobel derivatives (Sobel and Feldman, 1968). For every point in the edge map, all points which lie inside a maximum and minimum distance and lie along the gradient are incremented in the accumulator. Since the edge gradient in a circle always points towards the center, the incremented points are candidates for possible circle centers. The edge points for each circle center are also remembered.

Next, the algorithm selects all center candidates passing a threshold and which have no neighbours with a higher accumulator value. The selected centers are then sorted by amount of votes in descending order. Beginning at the highest supported center, the algorithm considers all points which were remembered as possible edges for this center candidate. The radius which is best supported among these points is then chosen, provided the support passes a threshold value and the radius is inside the accepted range. The remaining circles are detected in the same way.

In this way, circles can be found efficiently provided the various threshold parameters are sufficiently strict. For instance, if the radius is known, the runtime can be greatly decreased by posing strict restrictions the upper and lower radius bounds. For the wind turbine inspection scenario this method would be relevant for detecting the hub, which appears circular when viewed from the front.

The Hough circle transform is available in OpenCV as *HoughCircles()*.

4.3 Edge Detection

Edge detection algorithms are among the fundamental tools in the field of image processing. Their aim is to detect boundary paths in the image separating segments which display different pixel intensity. For the purpose of this paper, the main interest of an edge detector lies in that it provides requisite groundwork for the Hough transform (section 4.1).

A human contemplating an image usually has a clear notion of what to characterize as an edge. However, describing the theoretical properties forming the edges can prove less intuitive due to their variety. Edges most commonly appear as a change between one segment to another, as for instance when one object is covering another. Lines are another form, often referred to as a ridge, which may or may not be detected as two separate edges (one along each side) depending on the chosen approach. Lastly, there are corners, or junctions, referring to areas where two edges or lines cross. At these areas the directional rate of change properties of one edge is influenced by the other, which may pose both detection and localization issues for the algorithm. Furthermore, edges can vary from sharp edges with rapid jumps in intensity on one end to diffuse edges on the other end. This signifies that the scale at which the algorithm operates will influence its sensitivity.

Generally, edges are detectable through their spatial change in intensity in an image, and consequently most edge detection algorithms utilize some form of directional differentiation in their approach. Because of the noise intensifying properties of differentiation, some kind of noise reduction may be necessary to be applied beforehand, like for instance the Gaussian blur (section 4.6).

A large amount of differentiation operators have been designed to best capture specific kinds of edges, usually limited to first or second order directional derivatives. Among the most versatile is the Canny edge detection (Canny, 1986) which is well known for its accurate performance, and will for this reason be the edge detector of choice for the wind turbine tracking program.

4.3.1 Canny edge detection

The Canny edge detection (Canny, 1986), despite its early invention, remains among the most powerful edge detection algorithms. It is implemented in OpenCV as the function *Canny()* and will be explained in this section, yet many other implementation variations exist.

The first part of the algorithm consists of calculating the directional derivatives. This is done using the Sobel operator (Sobel and Feldman, 1968)

which utilizes two 3x3 kernels, one for horizontal and one for vertical differentiation, depicted by the matrices in equations (4.4) and (4.5). The kernels are convolved with the original image, such that differentiation is performed at every image point.

$$\mathbf{dx} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * Img \tag{4.4}$$

$$\mathbf{dy} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * Img \tag{4.5}$$

This results in the matrices **dx** and **dy** which contain directional derivatives along the x and y axis respectively. Using equations (4.6) and (4.7) the result is mapped to magnitude and direction form.

$$\mathbf{G} = \sqrt{\mathbf{dx}^2 + \mathbf{dy}^2} \tag{4.6}$$

$$\mathbf{\Theta} = arctan(\mathbf{dx}, \mathbf{dy}) \tag{4.7}$$

In this context, it is irrelevant whether the edge indicates a positive or negative change in pixel intensity. Therefore the range of the direction (for a specific point (x,y)) is limited to $\mathbf{\Theta}_{(x,y)} \in [0, \pi)$.

At this point, the algorithm is ready to continue to the second part which is non-maxima suppression. In this method the edges are thinned by keeping only the strongest gradients along the edge. To achieve this, one begins by defining four different direction angles: 0, $\frac{\pi}{4}$, $\frac{\pi}{2}$ and $\frac{3 \cdot \pi}{4}$. Each direction in the $\mathbf{\Theta}$ matrix is next rounded to the nearest of these angles. The algorithm now begins tracing each edge beginning in one of the four groups (note that the edge is perpendicular to the gradient direction). If along the edge, a neighbouring point in the gradient direction is of higher brightness, the current point is suppressed by setting its value to zero. When all points in this angle group have been examined in this fashion, the same procedure is performed on the remaining groups. Eventually, every edge is reduced to a width of only one point, where each point contains the highest gradient magnitude of the local cross-section of the original edge.

The last part involves converting the result to a binary edge image and removing weak edges which are not likely to be of significance. This is done through hysteresis thresholding which incorporates two threshold values in the following manner. A *high* threshold is used as a lower bound to initially

accept a point in an edge. When this point is accepted, however, neighbouring points on the edge may also be accepted if they pass a *low* lower bound. As a result, points with a weak gradient magnitude can be accepted as edge points if they are supported by stronger points elsewhere on the same line.

The Canny edge detector is available as *Canny()* in OpenCV, and is used to provide the necessary edge map for the Hough transform.

4.4 Corner Detection

Along with edge detection methods, the closely related corner detection methods provide another set of tools for feature detection. While edge detection methods are concerned about the rate of change in image intensity, corner detection algorithms extend this idea by examining how these edges curve in the image frame. Such curves can be exhibited by various features in the image. Some of which include intersecting edges, the ends of ridges, simple points and actual corners (sharp localized curvature of edge directions). Because of this diversity in appearance of corners the results from corner detection methods vary depending on what approach is used.

Among the most common methods are the SUSAN corner detector (Smith and Brady, 1997) and the Harris corner detector (Harris and Stephens, 1988). Zou et al. (2008) compared these detectors and concluded that the Harris corner detector performed better on the whole. Especially execution time was significantly quicker when running Harris, and will therefore be the choice for corner detection in this work.

The Harris corner detector works by first defining a weighted window $\mathbf{w}(x, y)$. For each point in the image, the variance in intensity, $\mathbf{I}(x, y)$, is found by comparing the neighbouring points inside this window. Neighbouring points are expressed as displacements, (u, v), resulting in the following:

$$\mathbf{E}(u, v) = \sum_{x,y} \mathbf{w}(x, y)[\mathbf{I}(x + u, y + v) - \mathbf{I}(x, y)]^2 \tag{4.8}$$

Harris and Stephens (1988) further showed that using Taylor expansion this can be approximated by (4.9), where $\mathbf{I}_x, \mathbf{I}_y, \mathbf{I}_{x,y}$ denote the directional derivatives of $\mathbf{I}(x, y)$.

$$\mathbf{E}(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} \mathbf{w}(x, y) \begin{bmatrix} \mathbf{I}_x^2 & \mathbf{I}_{x,y} \\ \mathbf{I}_{x,y} & \mathbf{I}_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.9}$$

From (4.9), one can define the Harris matrix, $\mathbf{M}$.

$$\mathbf{M} = \sum_{x,y} \mathbf{w}(x,y) \begin{bmatrix} \mathbf{I}_x^2 & \mathbf{I}_{x,y} \\ \mathbf{I}_{x,y} & \mathbf{I}_y^2 \end{bmatrix} \tag{4.10}$$

The intensity variance can thus be quantified by the eigenvalues of $\mathbf{M}$, but they are however not calculated individually. Instead the method utilizes the fact that $\lambda_1\lambda_2 = det(\mathbf{M})$ and $\lambda_1 + \lambda_2 = trace(\mathbf{M})$ to define a value R:

$$R = det(\mathbf{M}) - k(trace(\mathbf{M}))^2 \tag{4.11}$$

In order for the first term of R to dominate, it needs two high eigenvalues. This signifies presence of high variance of intensity in two directions, i.e. a corner. Corners can thus be identified where values of R bypass an imposed threshold.

The Harris corner detector is available in OpenCV as *cornerHarris()*.

4.5 Pinhole Camera Model

In computer vision, a major limitation is the loss of the third dimension when the world is projected to the image plane. However, some of this information can be restored if this mapping is known. The pinhole camera model describes this relation as if an ideal pinhole camera is used. Although it neglects some effects from lens cameras such as distortion, it still provides decent approximation for most quality cameras.

The ideal pinhole camera is modelled with an aperture where all light passes through a single point before hitting the image plane (figure 4.4). Thus each point is simply mapped by following a line through the pinhole, and equation (4.12) is obtained.

$$-\frac{z^i}{f} = \frac{z^c}{x^c} \tag{4.12}$$

The same applies along the y-axes.

$$-\frac{y^i}{f} = \frac{y^c}{x^c} \tag{4.13}$$

After modifying to get correct image orientation the final model is obtained:

$$\begin{bmatrix} y \\ z \end{bmatrix}^i = \frac{f}{x^c} \begin{bmatrix} y^c \\ z^c \end{bmatrix} = \begin{bmatrix} 0 & \frac{f}{x^c} & 0 \\ 0 & 0 & \frac{f}{x^c} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}^c \tag{4.14}$$

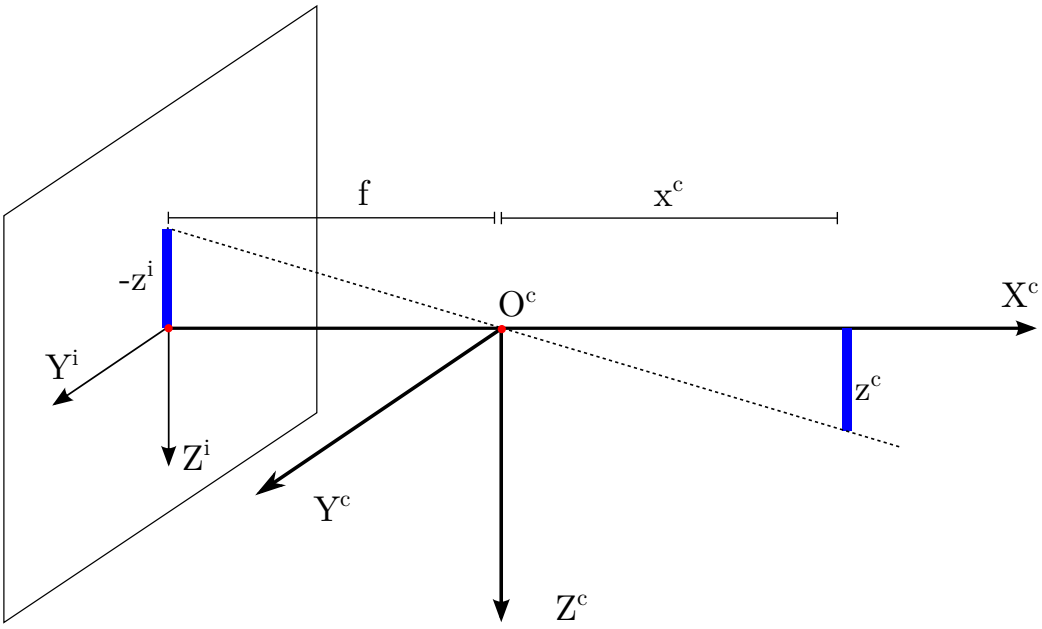


Figure 4.4: Geometry of the ideal pinhole camera.

The focal length, f, is the distance from the pinhole to the image plane. For a lens camera, the equivalent parameter can be found by taking multiple photos of an object of known size at different known distances. The result is then obtained from (4.14) by comparing the size at which the object appears in the images.

With the focal length known, one can utilize (4.14) to for instance estimate distances to objects or their size if one of those properties is known beforehand.

4.6 The Gaussian Pyramid and Gaussian Blur

The resolution of an image has a direct impact on computation time for most of the basic image processing functions. Hence, reducing the resolution is often the easiest and most effective way to reduce computational costs. Since the UAV on which the program is developed for will have limited processing power it would be preferable to keep the resolution as small as possible without prohibiting a usable result.

This part can be omitted if a camera which can readily stream images at an acceptable resolution is chosen. Otherwise the method halves the image resolution incrementally until it becomes lower than a user specified threshold.

The challenge of downsampling resides in deciding how to reduce pixel resolution while simultaneously minimizing change in visual appearance. Usually this is done by using image pyramids. An image pyramid refers to a representation where an image is repeatedly filtered and downsampled, in which the filtering method defines the type of image pyramid. For this program, a Gaussian image pyramid is used, elaborated by Birt (1981). The Gaussian filter is a smoothing (low-pass) filter where every pixel has its new value given as a weighted average of its surrounding pixels. In this particular case the kernel given by (4.15) is used, indicating how the surrounding pixels are weighed centered on the pixel being calculated.

$$\mathbf{H} = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 16 & 24 & 36 & 24 & 16 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \tag{4.15}$$

The downsampled image is the obtained by selecting every other pixel from the smoothed image. Gaussian blur is performed in the same way except the smoothed image remains at the same resolution.

Downsampling using the Gaussian pyramid is available through the function *pyrDown()* in the OpenCV library, while Gaussian blur is available is *GaussianBlur()*.

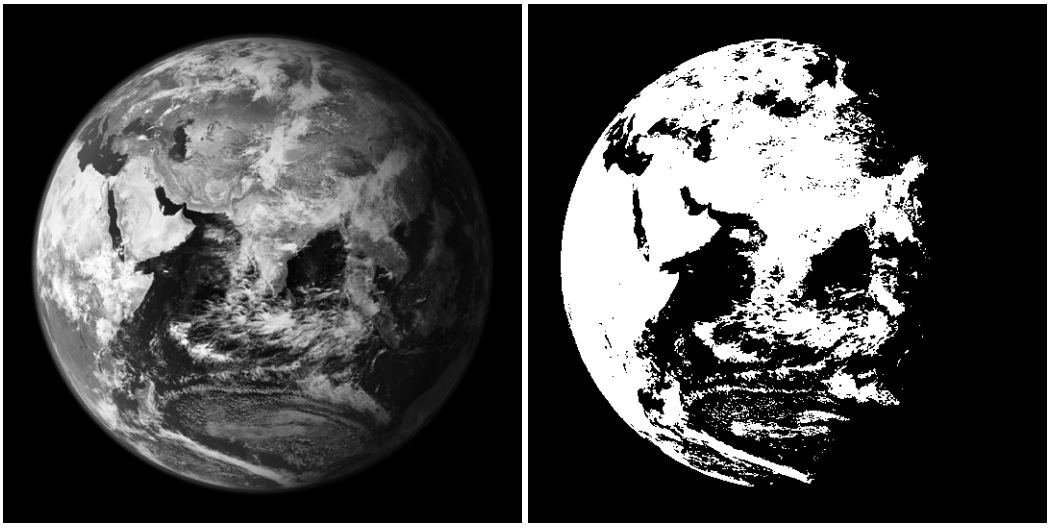
4.7 Multiband Thresholding and the HSV Color Space

Thresholding is a segmentation technique that outputs a binary image based on which pixels pass a threshold value. For a grayscale image this corresponds to keeping the pixels which are above a certain intensity, as defined in (4.16). This is illustrated by figure 4.5.

$$\mathbf{dst}(x, y) = \begin{cases} 1 & \text{if } \mathbf{src}(\text{x,y}) \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

One can convert images from one color space to another by using the function *cvtColor()* from the OpenCV library.

If the image is represented in a color space (such as RGB) the method must be augmented to include all channels. Hence, a separate threshold value is added for each parameter, thus creating the multiband thresholding



(a) (b)

Figure 4.5: (a) A grayscale picture of the earth. Source: NASA (2002). (b) The resulting output after thresholding the image. Only the brightest pixels which are above the threshold constitute the final shape.

method. In addition, an upper bound is added in order to permit an arbitrary range to be chosen. The resulting method is expressed in (4.17)

$$\mathbf{dst}(x,y) = \begin{cases} 1 & \text{if lowerR} \leq \mathbf{src}\text{(x,y,R)} \leq \text{upperR} \\ & \text{and lowerG} \leq \mathbf{src}\text{(x,y,G)} \leq \text{upperG} \\ & \text{and lowerB} \leq \mathbf{src}\text{(x,y,B)} \leq \text{upperB} \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

The wind turbine, however, has a color distribution for which BGR is not the most preferable color space. As mentioned in section 3, no particular color should dominate the tone of the wind turbine, and thus BGR has no apparent advantage in terms of object recognition.

Instead, the HSV color space is better suited for this purpose (Smith, 1978). It consists of the channels hue, saturation and value, which can be represented in a cylindrical coordinate system, as seen in figure 4.6. The hue parameter defines what one would call the pure color, e.g. blue, green and magenta. How intense these colors appear is given by the saturation parameter. This works in such a way that reducing the saturation to a minimum would convert the image to a grayscale image. Lastly, the value parameter can be equated to brightness. Increasing value reduces the amount

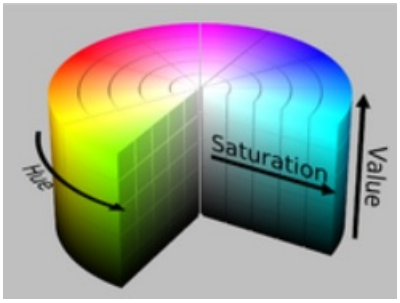


Figure 4.6: Cylindrical representation of the HSV color space (SharkD, 2010).

of shade in the color, but does not add white. The result is that saturation and brightness, which are the properties of interest of the wind turbine in terms of colors, can be easily manipulated. Since the wind turbine is color neutral, the wind turbine could be segmented by including areas which lack color and find appropriate saturation and value ranges.

4.8 Kalman Filter

The well-known Kalman filter (Kalman, 1960) has proved itself useful in a tremendous amount of applications, and numerous extension and variations of the Kalman filter have been developed.

The method works by utilizing measured data along with a system model to estimate the states of the system. By modelling system and measurement noise, the algorithm calculates the statistically optimal solution from the provided information.

The Kalman filter is designed by first defining the system model as follows:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \tag{4.18}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{4.19}$$

where $\mathbf{x}_k$ is the system states, $\mathbf{F}_k$ is the transition matrix, $\mathbf{u}_k$ is the control-input vector, $\mathbf{B}_k$ is the control-input model, $\mathbf{w}_k$ is the process noise, $\mathbf{z}_k$ is system measurements, $\mathbf{H}_k$ is the observation model mapping $\mathbf{x}_k$ to $\mathbf{z}_k$, $\mathbf{v}_k$ is the observation noise.

The process noise and observation noise are modelled by the process covariance matrix, $\mathbf{Q}_k$, and the measurement covariance matrix, $\mathbf{R}_k$, as follows:

$$\mathbf{w}_k \sim N(0, \mathbf{Q}_k), \quad \mathbf{v}_k \sim N(0, \mathbf{R}_k)$$

For each iteration of the Kalman filter loop, the following steps are performed in the given order

Prediction of states and covariance matrix

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F}_k \hat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{B}_k \boldsymbol{u}_k \tag{4.20}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^T + \boldsymbol{Q}_k \tag{4.21}$$

Calculate Kalman gain

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1} \tag{4.22}$$

Update states and covariance matrix

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k (\boldsymbol{z}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_{k|k-1}) \tag{4.23}$$

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k|k-1} \tag{4.24}$$

A computer vision program is usually strongly exposed to noise. The camera itself can be a significant source, and (especially outdoor) environments are of constant change. Furthermore, flying with an UAV causes vibrations and quick movements. All these are factors which will produce frame-to-frame inconsistency in the results of the computer vision methods, which makes utilizing the Kalman filter very attractive. In addition, the predictions from the filter can be utilized to keep track of an object even when no measurement is made.

Chapter 5

System Overview

This chapter offers an overview over both hardware and software in the system. A base set-up common for the whole student team working with hexacopters was provided at the beginning of this project. Only video camera and software was further altered in the project.

5.1 Hardware Setup

The system topology is shown in figure 5.1. The PandaBoard is the core of the system, where the computer vision software is run, implemented in the Dune environment. From USB connection Dune has access to the video camera, in addition to IMU and GPS data from the ArduPilot board. During flight a computer may send commands and supervise via Neptus over a WiFi connection. Direct control of the hexacopter via radio control is supported by the ArduPilot. A description of the individual components follows.

5.1.1 Hexacopter, Arducopter 3DR Hexa B

The Arducopter 3DR Hexa B (Robotics, 2014) was used for the project. It is composed of six aluminum arms, and fiberglass boards and landing gears, ensuring a both lightweight and robust structure. With the six motors it is capable of lifting over 1 kg of additional payload. The hexacopter was designed and manufactured by 3D Robotics. Figure 5.2 shows the hexacopter along with carried payload.

5.1.2 ArduPilot Mega 2.5/2.6

The ArduPilot Mega (APM), is an open source autopilot system. It is equipped with an IMU providing 3-axis gyros and accelerometers. Addition-

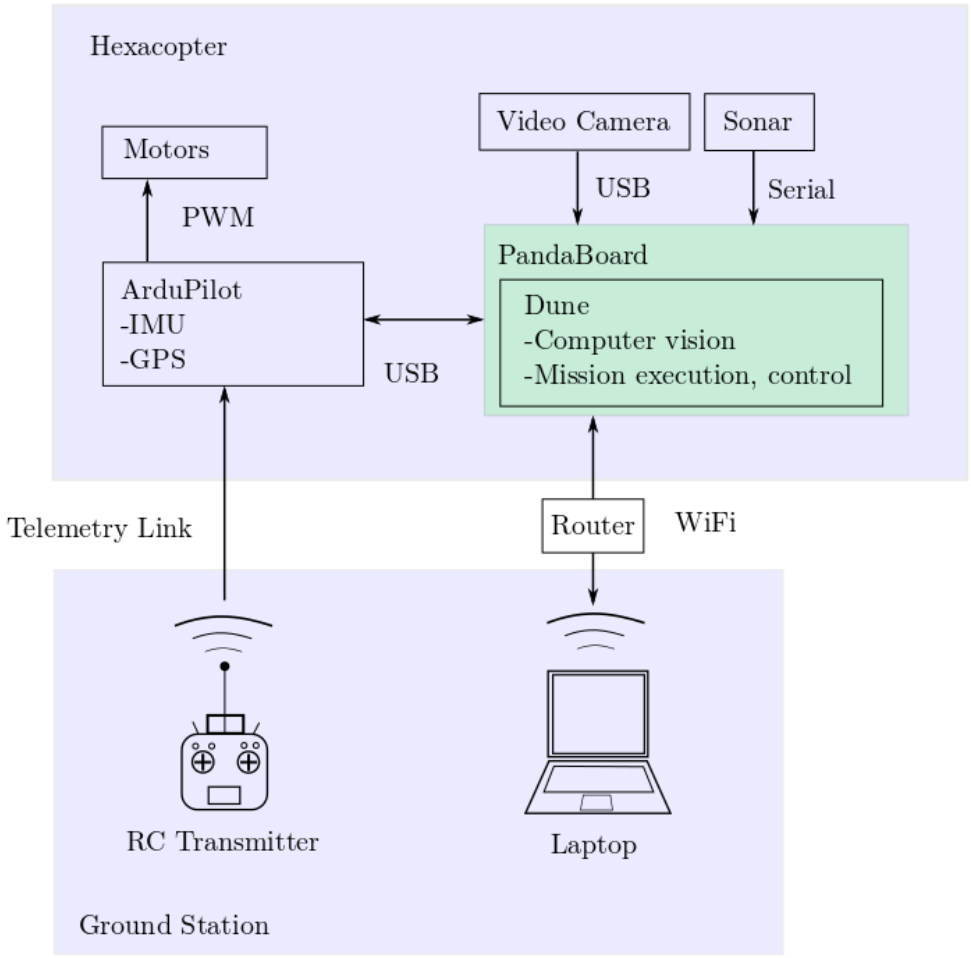


Figure 5.1: System overview, showing how the various hardware components are integrated.

Figure 5.2: The hexacopter as configured for the project, with payload. Photo: Thor Audun Steen

Table 5.1: Specifications for PandaBoard ES

Processor	Dual-core ARM Cortex-A9 MPCore, 1.2 GHz
Memory	1 GB DDR2 RAM
Weight	81.5 g
Dimensions	114.3 mm x 101.6 mm
Power Requirement	5V/1A

ally, it carries altimeter, GPS, interface to motors, telemetry link and USB for external connectivity. Together with control software it allows the user to operate the hexacopter from radio control and also supports GPS missions via waypoints. The open source code can be altered for specific needs, and the USB support is used to connect the more powerful PandaBoard. For the connection between PandaBoard and APM, the MAVLink protocol is used.

5.1.3 Single-Board Computer, PandaBoard ES

A single-board computer (SBC) is a board comprised of all necessary components required to function as a computer. The task of the SBC in this system is to run the control and computer vision software, and to provide access to periphery components through its I/O interface. The Pandaboard ES (PandaBoard, 2014) is the SBC of choice, by recommendation of Leira (2013). Specifications are shown in table 5.1. An overview of the PandaBoard is shown if figure 5.3.

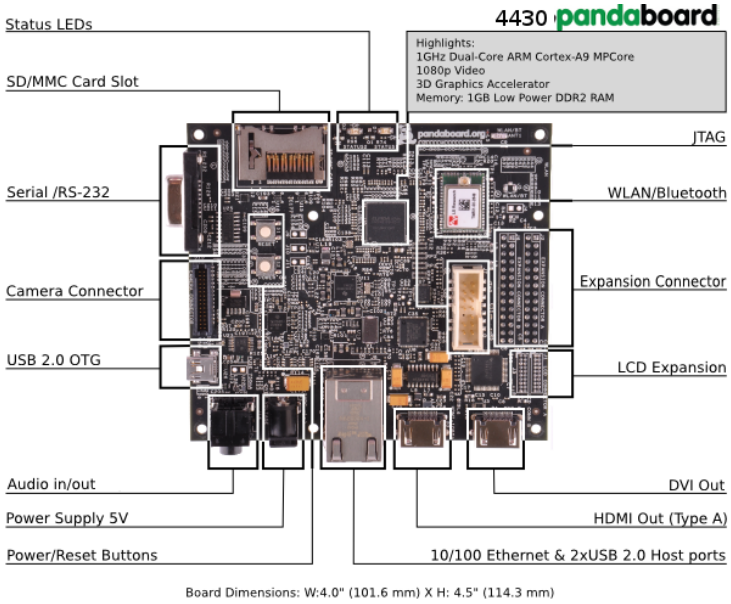


Figure 5.3: Overview of the PandaBoard. Source: PandaBoard (2014)

5.1.4 Video camera

As this work is heavily focused on computer vision, the video camera is perhaps the most important component in the system. The camera should provide decent image quality, while as all components in general, be small in weight and size. Fortunately, by the current state of technology this is not an issue seeing high speed and high resolution video cameras by the size of a thumb are available. Furthermore, the computational demands of image processing methods usually increase with resolution, requiring resolution to be limited anyway.

Because the line detection plays a major role in the computer vision program, it is desirable to have a minimal amount of distortion in the image. Distortion is problematic as it causes straight lines to bend, which may leave them undetected or segmented. It is a radial effect which becomes more prominent towards the edges of the image. Since the blades of the wind turbine are expected to stretch past the edges of the image, distortion is expected to have significant impact. It should be avoided altogether by using a camera where distortion is not prevalent.

Regarding interface, OpenCV provides uncomplicated support for the Video4Linux (V4L) driver framework which is closely integrated with the Linux kernel. V4L supports most USB cameras, which makes using a USB camera a natural choice.

5.1.5 e-CAM51_USB

The e-CAM51_USB camera was used for testing in this work. It is small and light, which makes it suitable for UAV applications. The shape, however, made it somewhat impractical to mount (see figure 5.4). It is connectible by USB, and supported configuration via V4L, giving control over parameters such as autofocus and exposure time. Most importantly it could be configured to provide lower resolution video to reduce computation demands for the image processing methods. No notable distortion was observed using the camera. Specifications are given in table 5.2.

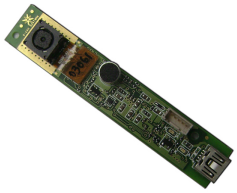


Figure 5.4: The camera. Source: e-con Systems (2014)

Table 5.2: Specifications for the camera

Video properties	
Max video resolution	1080p
Max frame rate	30fps
Field of view	60° Diagonal
Physical properties	
Dimensions	71mm×13mm×7.8mm
Interfacing	
Power requirements	5V/400mA
Video and power connection	USB

5.2 Software Overview

The program described in this report was written in the Dune environment, as a task named *Vision.WindmillTrack*. By the modular design of Dune, the task is simply included when needed in the system, and similarly other parts of the system, e.g. control tasks, are available for the computer vision program. These parts communicate via the IMC protocol. The implementation also utilized the OpenCV library to a high degree, because of the extensive

amount of image processing methods it provides. It all runs on the Ubuntu operating system and is installed on a memory card on the PandaBoard. The software can be supervised and given commands from a remote computer using Neptus, which also communicates via the IMC protocol. A brief description of the mentioned software follows.

5.2.1 Ubuntu

Ubuntu is a Debian-based open source operating system which was initially released in 2004. Over the years it has become fairly widespread and been provided continuous updates. Meanwhile, a notable development community has emerged, where advice and support can be received as more applications are investigated. Some of the merits of the operating system are its reliability and customizability. Customization allows the operating system to be tailored to the specific usage. For this purpose, this includes stripping off functionality which is not required, thus releasing resources to be focused on the essential tasks. In addition, Ubuntu is the operating system favoured by the PandaBoard development community, and thus there exist considerable amounts of helpful documentation for this set-up.

5.2.2 OpenCV

The Open Source Computer Vision (OpenCV) library was developed by Intel Russia research center aimed for real time computer vision applications. It contains a variety of algorithms ranging from the most basic image processing methods to more sophisticated implementations. Additionally, it contains core functionality including interface to video devices and a variety of structure definitions allowing convenient management of image arrays and other relevant entities. Furthermore, it provides means for quickly setting up a graphical user interface. All this adds up to a great tool for development which is both powerful and easy to use. OpenCV has also been implemented with emphasis on being computationally efficient, making it suitable for an on-board system where computation power is limited.

5.2.3 Dune

DUNE: Unified Navigational Environment is an on-board software solution for unmanned vehicles. The hexacopter will run on this software, and the presented program is written as an extension to this environment. The software solution provides means for interacting with the connected components as well as control, navigation, supervision and plan execution. Furthermore

it is both CPU architecture independent and OS independent. It is written in C++ and developed by *LSTS: Underwater Systems and Technology Laboratory*.

5.2.4 Neptus

Neptus is a command and control software operated from a ground station. It is designed to operate well together with Dune and was also developed by LSTS. Neptus provides tools for remotely monitoring UAVs and assigning plans and commands in real-time missions, supporting multiple connections dynamically. Furthermore, it provides possibilities for both simulating missions and reviewing previously performed operations. This is presented in a customizable interface equipped with map layers and control panels. It is written in Java and available for both Windows and Linux systems.

5.2.5 IMC

The *Inter-Module Communication* (IMC) protocol was developed by LSTS to provide reliable communication between the systems. The protocol is message-oriented, such that messages can be sent and received from a bus which connects independently run threads or systems. Thus it functions as a method of communication between tasks internally in Dune, and can also be passed to and from other vehicles or computers running Dune or Neptus. In this project specifically, the IMC bus provides the link between the computer vision task and the other rest of Dune, and also between the vehicle and any connected computer.

Chapter 6

Recognizing and Tracking the Wind Turbine

6.1 Recognizing the Wind Turbine

The act of object recognition refers to the task of detecting and identifying objects in an image. In most practical applications, one has limited the problem to detecting one or a specific set of objects, thus greatly simplifying the issue. In this case, the task is focused on attempting to detect the presence of a wind turbine. This section will investigate some approaches to detect characteristics in the image and discuss if they offer a reliable base for a complete detection, and describes the approach which was implemented in section 6.1.5. How the approach is implemented into the rest of the system is explained in chapter 8.

6.1.1 Feature detection using the Hough line transform

The Hough transform (described in section 4.1) is attractive because the produced lines offer very practical material for further analysis. An example is show in figure 6.1. Figure 6.1a shows an image taken at Valsneset wind farm. The binary edge map in figure 6.1b is the output after performing the Canny edge detection on the original image. One can see the contour is clearly extracted. The result from the Hough transform is shown as the blue lines in figure 6.1c, which match the contours from the edge map tightly.

Considering the detected Hough lines, there are some lesser issues which will be addressed. Sometimes a line which optimally should be detected as a single line is broken into smaller segments. This is apparent at for instance

(a) (b)

(c)

Figure 6.1: (a): An image of a wind turbine taken at the wind farm at Valsneset, Norway. (b): The binary edge map obtained from running the Canny edge detector on the raw image (a). (c): The result after performing the Hough transform using the edge map in (b).

the lower left blade in figure 6.1c, where due to the slightly bent edge at one side of the blade the edge is detected as two separate and almost connected lines.

Another issue may appear if there is a significant amount of noise in the image. Small changes can affect the resulting edge map, resulting in frame to frame varieties of the Hough transform. This has specifically a notable effect of the start and end points of the detected lines.

Those matters aside, the Hough transform provides an accurate description of the contours of the wind turbine. The direction and location of the lines makes it a simple matter to identify the blades and tower along with their orientation. One approach for this could be to detect the vertical tower lines and then searching for blade lines intersecting the top. If this approach yields a match, the probability is high of having found an actual wind turbine, due to its distinct shape. In addition as described in section 4.1, the Hough transform version which is available as *houghLinesP()* in OpenCV runs relatively efficiently. These properties make the Hough transform attractive, and the Hough transform was therefore chosen as the basis the recognition program. The approach mentioned earlier in this paragraph will compose the remaining steps of the wind turbine recognition and is explained in detail in section 6.1.5.

6.1.2 Feature detection using corner detection

Corner features in an image often indicate particularly interesting points. An example is shown in figure 6.2 where the Harris corner detector was used. One observes that some of the essential points are indeed found, and one could image to connect the dots and obtain a good description of the object.

A problem arises, however, when other objects which exhibit strong features are present. Then, one does not know which detection belongs to what object. By comparison, the Hough transform is different in this regard, since the lines are organized in a pattern where orientation and end points can be related to the whole shape.

Furthermore, there is more uncertainty related to the corner detection. As seen in figure 6.2, for instance the tip of the lower blade and the sharp corner below the hub are points which were expected to be detected, but were not. They could actually be detected by increasing the sensitivity, but the problem remains to find this sensitivity. The optimal input parameters for the algorithm where found to vary significantly depending on image properties, specifically lightning and noise.

A suggestion could be to use corner detection in conjunction with a method such as the Hough transform, such that the detected points are

Figure 6.2: The red circles show the detected corners using the Harris corner detector on the image from figure 6.1a.

easier to identify as a part of the whole. In this way, one could apply the corner detection to more accurately locate points of interest and thus provide means for more exact measurements of lengths in the image.

6.1.3 Detecting the hub using Hough circle transform

The circular shape of the hub could be a suitable target for the Hough circle transform. If a circle was successfully detected enclosing the hub, the hub position could by estimated from the circle center position.

An example from a result using the circle transform is shown in figure 6.3. The figure shows that one of the circles encloses the hub as desired, but similarly to the corner detection there is uncertainty regarding the correctness of the result. One does not know whether or which circle resembles the hub.

For this reason, this method was not utilized in the recognition program. However, as will be mentioned later, for further development of the program it is suggested to investigate the method further. One approach could be to limit the radius bounds to a tight range, where the radius bounds match the expected size of the hub in the image based on a distance estimation. Another application could be to detect the hub by searching for a Hough circle centered along the direction of the tower.

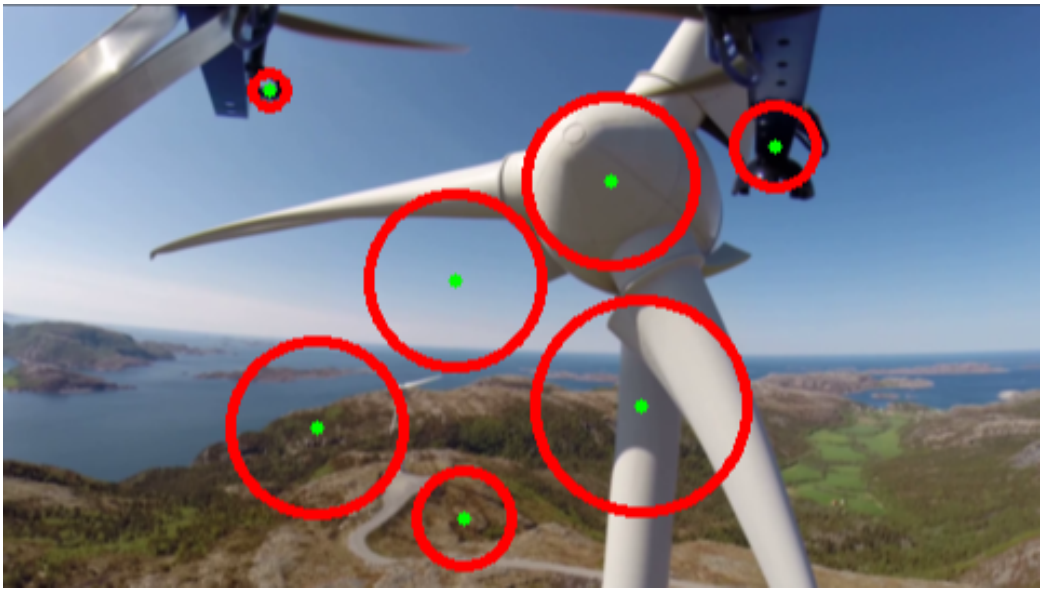


Figure 6.3: The red circles, centered at the green dots, show detected Hough circles. One of the circles encloses the hub as desired.

6.1.4 Image segmentation

Another way of identifying an object of interest is through image segmentation. This refers to dividing the image into several segments such that the content of each segment possesses a certain characteristic. In the case of the wind turbine, it is recognizable by its homogeneous gray color.

Among the most basic image segmenting algorithms are the color thresholding methods (section 4.7). Figure 6.4 shows a result of running the multi-band thresholding method using the HSV color space. In figure 6.4a the thresholding bounds are tuned to properly extract the wind turbine from the image. From such a distinct segmentation there are many ways to further inspect the wind turbine. For instance, a border following algorithm such as presented by Suzuki and Abe (1985), would identify the contours of the wind turbine. Further one could utilize convexity analysis and identify the convex hull which is the minimal convex set containing all the points of the contour. The corners of the convex hull would thus identify the tip of the blades and tower, while the hub would lie close to the points furthest away from the edge of the convex hull.

Obtaining such a segmentation of the quality as depicted in figure 6.4a is hard without manual tuning. The threshold bounds need not be far off before the segmented image becomes drastically less useful, as shown in figure 6.4b. The difference from 6.4a to 6.4b is merely the upper bound for the value

(a) (b)

Figure 6.4: The results from running color thresholding on 6.1a. (a): Thresholding bounds are tuned for optimal segmentation. (b): Thresholding bounds are not tuned well.

parameter (V in the HSV color space) has been increased from 95 to 125 (the range is 0-255). Determining the optimal bounds is a tough task as the result can vary strongly with lightning conditions and the background. Therefore the approach is deemed too unpredictable for this purpose.

A more sophisticated approaches could for instance include an adaptive scheme for thresholding. However, since the thresholding process is relatively slow, an adaptive approach utilizing feedback from several iterations would require excessive computation demand.

6.1.5 Recognizing the wind turbine from detected Hough lines

This section explains how recognition was implemented based on the results from the Hough line transform. A step by step description is given to clarify the process.

6.1.5.1 Locating the tower

Search for vertical lines

Among the parts of the wind turbine, the tower is arguably the most certain to correctly identify due to its consistency in appearance. Similarly to the blades it tends to exhibit long and distinct edges. But whereas the blades are free to settle in any orientation, the tower is fixed in its vertical stance. Therefore one could begin by searching for vertical lines, with a few degrees tolerance.

Compensate for roll

An issue arises due to the prevalent roll rotation of a hexacopter. Naturally, this translates to a corresponding rotation of the image frame, and a counteraction must be made. One solution which was implemented is to use the current estimated roll orientation from the IMU to adjust the search angle. Another option is to mount the camera on a gimbal stabilizer to physically prevent rotation of the image frame.

Restrict end points of lines

Because the UAV expectedly flies at an altitude at level with the hub, the base of the tower will be below the bounds of the image and consequently the lines exhibited by the tower span beyond the lower edge of the image frame. Based on this knowledge another restriction is imposed; the lines representing the tower must possess an ending point near the lower bound of the image.

6.1.5.2 Locating the blades

If at least one line was identified as part of the tower, the program proceeds to search for the lines exhibited by the blades. By locating the tower, a much better evaluation can be made on whether a line belongs to a blade, by using their relative geometric properties.

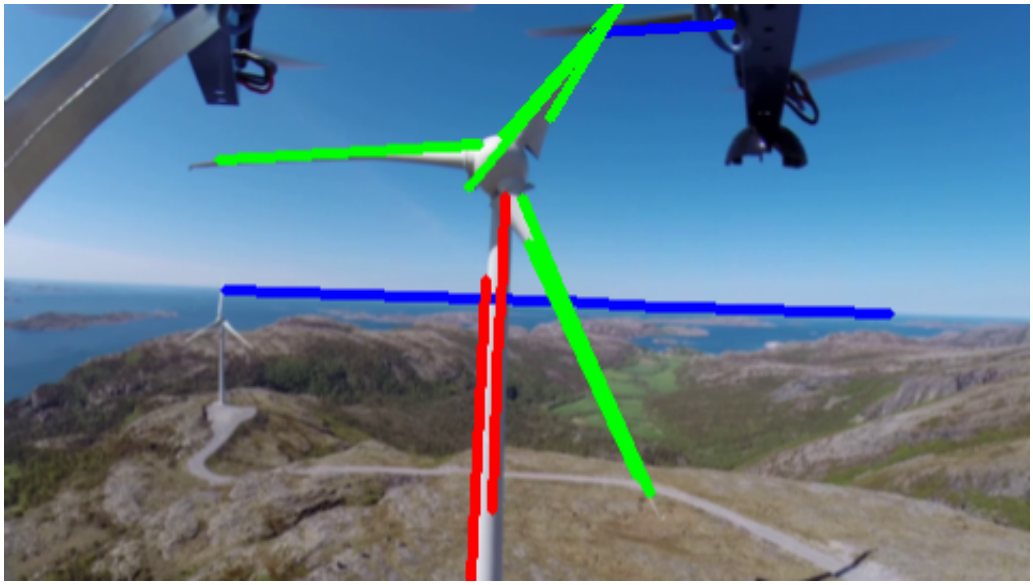


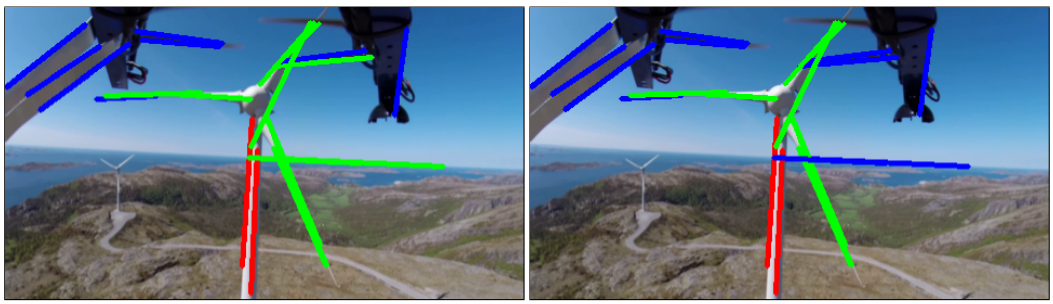
Figure 6.5: Example on how Hough lines get classified as tower lines (red) and blade lines (green).

Search in vicinity of estimated hub position

By identifying the highest point among all the tower lines an estimate is provided for the hub. Thus by searching for lines in a circular area around this point one can expect to find lines belonging to the blades. In specific, the search is conducted for ending points of lines, such that lines which start or end close to the estimated hub center are found, but lines which pass through the area are ignored. Lines which already where identified as tower lines are ignored.

A search radius which worked well at about a 30 meter distance was 0.3 times the image height. Using the pinhole model this search radius can be adjusted for other distances by equation (4.14)). This can be done by using the radius as z^i, and then change the distance, x^c, and find the matching z^i which keeps the expression constant. Due to uncertainty in distance measurement, this approach for search radius adjustment was not implemented to run dynamically but rather set manually prior to detection.

A typical issue with this approach occurs when the horizon appears at a height level close to the hub, which can potentially lead to it being incorrectly identified as a blade. Unfortunately, one cannot simply ignore horizontal lines, because (as is apparent in figure 6.5) a blade may also be horizontally aligned. But since the horizon most of the time lies some distance below the hub, the issue was disregarded. The implications are discussed in section



(a) (b)

Figure 6.6: (a): The green lines correspond to the initially detected blade lines. There are false detections caused by the horizon and the UAV. (b): The false detections have been removed by the voting procedure. Only the green lines corresponding to the actual blades remain.

10.1.

Orgranize the blade lines

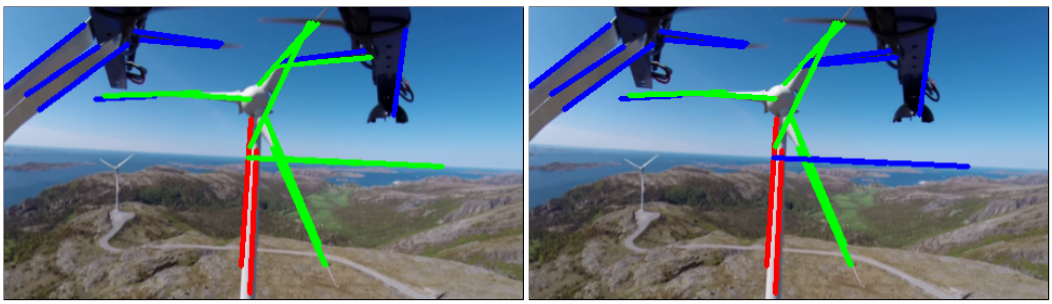
The lines which were determined to belong to blades are sorted by angle and grouped with other lines of adjacent angles. Together each group form a single blade object, where their average angle determines the estimated angle of the blade. The tower lines similarly form a tower object.

Eliminate false blade detections

Lines which were missed in tower detection could later mistakenly be detected as blade lines. Therefore, a blade is removed if has been detected with an angle pointing downwards such that it overlaps the tower angle.

Furthermore, the property of there being a 120 degree angle between the blades was utilized by implementing a voting system. A blade which receives a certain amount of votes is assumed to be a false detection and is removed. The voting is conducted by comparing each detected blade to each other. If the angle between two blades is not close enough to 120 degrees by a certain tolerance, both blades obtain a vote. The tolerance was set to 20 degrees to accommodate for measurement error and skewed angles due to potential yaw angle between the UAV and wind turbine.

Setting the vote limit to 3, such that three votes or more signifies a false blade, proved to work well. Thus, if for instance, the three actual blades plus a fourth false blade is detected, then the false blade receives three votes in total from the actual blades and is removed. Meanwhile, the true blades



(a) (b)

Figure 6.7: (a): The green lines correspond to the initially detected blade lines. There are false detections caused by the horizon and the UAV. (b): The false detections have been removed by the voting procedure. Only the green lines corresponding to the actual blades remain.

receive only one vote from the false blade. Clearly a higher limit would not remove the false blade in this case, and would defeat the purpose of the voting. Furthermore, the limit is high enough to provide small a buffer against mistakenly removing the true blades. Figure 6.7 shows how the false blade lines are removed when voting is successful.

6.1.5.3 Locate the hub center

Calculate intersections

Since all the blades beam out from the hub, the hub location can be found by calculating their common intersection point. However, can be observed in figure 6.5 the detected lines for the blades are sources of disagreement. For instance, by looking at the upper right blade one observes that the curvature on the lower edge causes a steeper angle on the corresponding line compared to the lines from the upper edge. Furthermore, since lines are detected at both edges of the limbs there will be a spatial disagreement as well. At this point it is unclear which lines provide the appropriate result. This issue was handled by calculating the intersection points between every line, except between lines belonging to the same limb (figure 6.8), then computing the average.

Obtain average intersection point

From the previous step there may appear some stray intersection points causing discrepant results. For instance, if the horizon is incorrectly detected as a blade it can intersect with the other blades at a distance far from the hub. To prevent such kinds of scenarios, restrictions are imposed on the points

Figure 6.8: The green circles show the points where the blade lines intersect each other.

of intersection. Points which have horizontal position far from the line extended by the tower are ignored. Since the actual vertical position of the hub is more uncertain, a slightly less strict restriction is imposed on vertical distance based on the temporary estimated hub center (highest point among the tower lines). Figure 6.9 show an example of how points are filtered via this approach. After the points caught by the restrictions have been removed, an average is calculated from the remaining points which should result in the final estimated hub center position, similarly to figure 6.10.

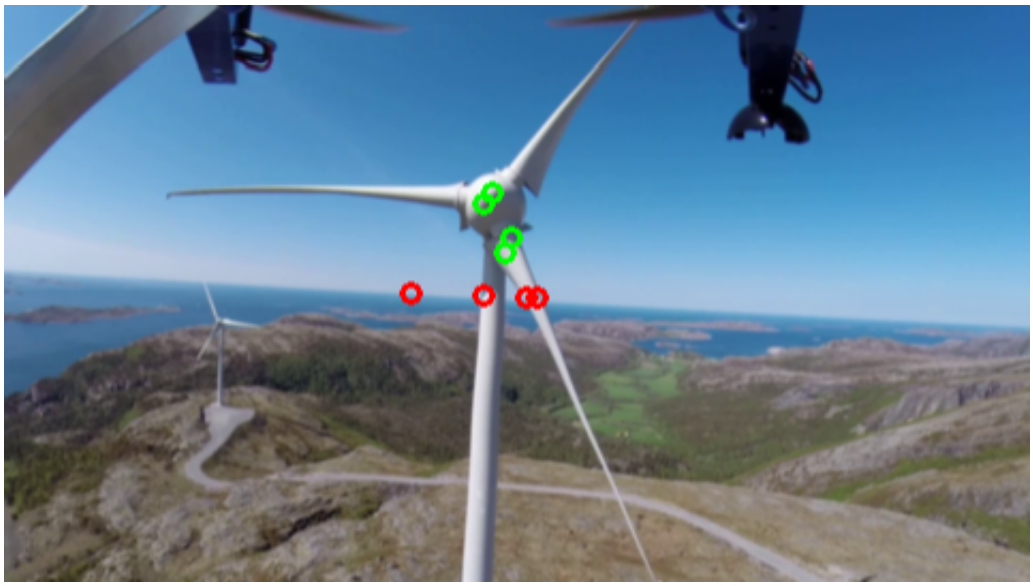


Figure 6.9: In this case there where some bad intersection points due to the horizon being detected as a blade line. The red circles show intersection points which were removed by posed restrictions, leaving the better green points for the calculation of average.

Figure 6.10: The red circle shows the estimated hub center position calculated from the average of the intersections points from figure 6.8.

6.2 Object Tracking

Object tracking in computer vision refers to gathering information on how an object moves or evolves over time in the image. This requires state information of the object being saved or updated between frames. Usually, these states contain position and its derivatives, although depending on the problem one might be inclined to track orientation, size, color or other properties.

For this program the interest lies in tracking the position of the hub center, so that the UAV can be maneuvered in the fashion described in section 7.1. In an object tracking routine one is commonly faced with two problems which prevent smooth tracking. Firstly, there is noise, which is an accumulated result of the collected sources of error from the camera and the various algorithms in the program. Secondly, there may occur frames where the tracked object is not detected. This can happen for instance if the objects disappears behind another object. For this report, a more relevant scenario is when others entities produce interfering Hough lines or if a crucial edge fails to be captured by the edge detector. Often this happens for several frames in succession, therefore some amendment should be devised for this event in order to prevent unpredictable behaviour.

Both of these problems can be dealt with using the Kalman filter (section 4.8). The noise is handled through filter's noise reducing properties. For the case when the object is not detected one may estimate its state using a prediction from the model of the filter. This is done by only running the prediction update in the Kalman filter, while skipping the other update steps. In other words, when no new measurements are made, it is assumed that the object strictly follows the model. Consequently, the program will still have knowledge of the states of the object, and can act accordingly.

Now, the implemented procedure will be described. First, a temporary state model of the hub center is defined as follows.

$$\underbrace{\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_k}_{\boldsymbol{x}_k} = \underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{F}_k} \underbrace{\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{k-1}}_{\boldsymbol{x}_{k-1}} + \boldsymbol{w}_k \tag{6.1}$$

Here the input term, $\boldsymbol{u}_k$, is disregarded. One could imagine using the movement of the UAV to describe its effect on the image, but this would result in an overly complex model which would also be heavily dependent on accurate distance estimation.

Equation (6.1) is simply a model where the x and y positions are described

by constant velocity:

$$\begin{aligned} x_k &= x_{k-1} + \dot{x}_{k-1} + w_{x,k} \\ y_k &= y_{k-1} + \dot{y}_{k-1} + w_{y,k} \end{aligned} \tag{6.2}$$

The measurements are the x and y positions of the hub center, as identified through the recognition part (section 6.1.5.3), leading to the following.

$$\underbrace{\begin{bmatrix} z_x \\ z_y \end{bmatrix}}_{\boldsymbol{z}_k} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}_{\boldsymbol{H}_k} \underbrace{\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}}_{\boldsymbol{x}_k} + \boldsymbol{v}_k \tag{6.3}$$

The result from this scheme will be that if the program fails to recognize the hub center from the image, it will assume that it follows constant velocity from its current position and velocity. However, if this goes on for a significant period of time, the hub center position will eventually continue moving past and beyond the edge of the frame. To counter this issue, damping is included into the model, which is done by altering the transition matrix, $\boldsymbol{F}_k$, to the following.

$$\boldsymbol{F}_k = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & \gamma_x & 0 \\ 0 & 0 & 0 & \gamma_y \end{bmatrix} \tag{6.4}$$

Where $\gamma_x, \gamma_y \in [0, 1]$ Leading to the x and y position evolving according to

$$\begin{aligned} x_k &= x_{k-1} + \gamma_x \dot{x}_{k-1} + w_{x,k} \\ y_k &= y_{k-1} + \gamma_y \dot{y}_{k-1} + w_{y,k} \end{aligned} \tag{6.5}$$

Which means that for each frame, only the magnitude specified by γ_x, γ_y of $\dot{x}_k$, $\dot{y}_k$ carries over to the next frame, such that $\dot{x}_k$ and $\dot{y}_k$ exponentially decay over time.

However, if the program the program is not able to locate the hub after an extended period of time, instead of continuing to blindly relying on the model it should switch to a searching state where movement is stopped until the wind turbine has again been identified (for instance by simply rotating around the yaw axis). Mainly because the issue is more likely caused by no hub in sight or other detection issues rather than an unlucky sequence of video frames.

Chapter 7

Maneuvering Plan for Wind Turbine Inspection

7.1 Maneuvering Towards the Wind Turbine

This section gives a suggestion for an approach on how to maneuver the UAV to reach the target destination. The target destination for the program documented by this report is a position about 5 meters in front of the hub of the wind turbine, starting from an initial position.

In this context, the initial position refers to the position of the UAV at the time when the computer vision program is activated. It is assumed that the GPS system is used to navigate to this position. Due to the limited GPS accuracy the target initial position (depicted by the red cross in figure 7.1) should be assigned at a safe distance from the wind turbine. For both simplicity and in order to avoid the blades of the wind turbine the target should also be assigned directly in front of turbine (wind turbine facing UAV). The actual initial position is expected to appear in a limited area around the target initial position, as illustrated by the blue area in figure 7.1. In this way, the program initiates at a position where it is ensured to be on the front side of the wind turbine and at an angle which captures the characteristic shape of the wind turbine.

The ultimate goal of the wind turbine tracking program will be to position the UAV at a distance of about 5 meters in front of the hub of the wind turbine. Naturally, there are numerous approaches on how to navigate to this position. The method which was designed and implemented in the program will now be presented.

The method can be viewed as three separate tasks (T1, T2 and T3). Each task is given a priority from T1 at highest to T3 at lowest. Only

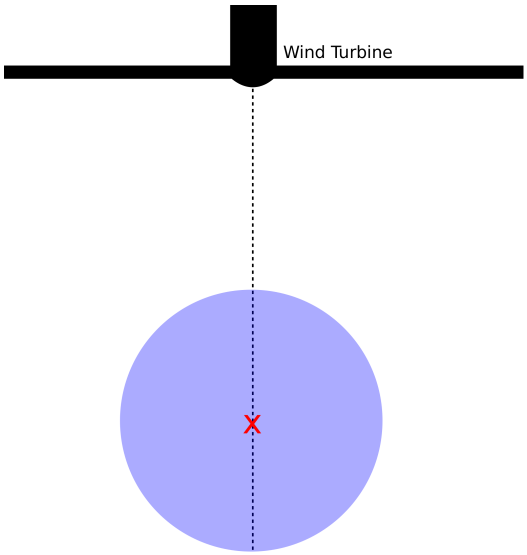


Figure 7.1: Top-down perspective of target initial position (red cross) in relation to the wind turbine. The blue area shows where actual initial position can be expected to appear.

when the conditions for one task is fulfilled the program moves to the next. Furthermore, if during one task the conditions for a higher prioritized task is invalidated the method reverts back to complete the former task. A summary of the procedure is given by algorithm 2.

The first task, T1, makes sure that the UAV is facing the target position i.e. the hub of the wind turbine. This is done by defining a rectangular boundary in the image frame in which the UAV attempts to keep the target, as illustrated in figure 7.2. If the target is inside the boundary, the conditions for this task are fulfilled and the next task is initiated. Else if the target is above or below the boundary, the UAV simply moves up or down to bring the target back in the frame. One could imagine also achieving this by altering the pitch. However, since the pitch is crucial to movement and stability of a multicopter it is not a viable option for this system. On the other hand, if the target is outside the horizontal bounds of the rectangle there are two choices of action; move the UAV laterally or rotate the yaw angle. Rotating is preferred for several reasons. First it requires less work, which also means that a rotation is more probable to have brought the target outside the boundary in the first place. Second, by assuming the UAV already has maneuvered to a position which minimizes the yaw angle relative to the wind turbine, repositioning the UAV instead of rotating it could cancel this by increasing the angle again. Instead, left and right movement is reserved

Figure 7.2: The red rectangle illustrates the boundary in which the UAV attempts to keep the target (blue dot). In this example it is clear that the target is above the upper bound, and the UAV will fly upwards as commanded by T1.

for the next task.

For the second task, T2, the objective is to minimize the relative yaw angle of the wind turbine (figure 7.7). This is done by measuring the yaw angle as described in section 7.4. By flying laterally, the relative angle is reduced until a certain tolerance is reached. Naturally, by laterally one will also experience the wind turbine moving horizontally in the camera frame, albeit slowly compared to the effect of UAV rotation. By consequence, once the target moves outside the rectangle described in the first task, the first task becomes prioritized and adjusts the attitude to bring the target back inside the boundary. The result of the combined alteration of T1 and T2 is thus a circular motion around the wind turbine, as illustrated in figure 7.3 by the segments marked T1 and T2. As the relative yaw is reduced past a certain threshold indicated by the green triangle, the relative yaw is deemed as sufficiently small and the method proceeds to the third task.

The third and last task, T3, simply commands the UAV to move forward towards the target until the desired distance has been reached. A correct course is ensured and maintained by the former tasks. Should a disturbance cause the UAV to lose its heading or get a too big angle, it reverts to T1 or T2, respectively, to correct the error. As the UAV gets close the desired distance can be detected using the camera or a sonar as described in section 7.3.

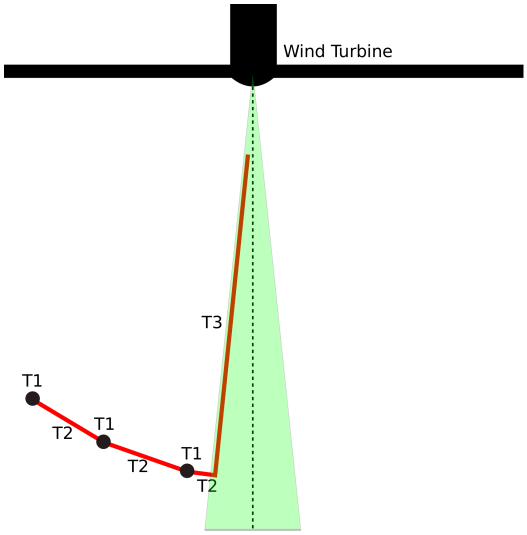


Figure 7.3: Top-down perspective showing an example scenario of the navigation path (red line), with a slightly exaggerated initial position. Indications show when each task is run. The green triangle depicts the tolerated range for relative yaw.

Algorithm 2 Navigation method

```
while not reached desire distance do
    if target not in boundary then                          ▷ T1
        if target above/below then
            move vertically
        if target too far left/right then
            rotate along yaw axis
    else if |relative yaw| > yaw tolerance then             ▷ T2
        move laterally
    else                                                    ▷ T3
        move forward
```

The result is an approach with a predictable behaviour which can be easily implemented. It also has the advantages of keeping the wind turbine in the field of view of the camera at all times and utilizes the available data in a simple and practical way. As seen in figure 7.3, the formed path keeps a safe distance from the blades. In addition, keeping the low yaw angle is beneficial for the computer vision part. A low angle also places the UAV directly in front of the wind turbine when the desired distance is reached, providing a favourable starting point for the subsequent blade inspection program.

Figure 7.4: An example on what the starting position might look like when the blade inspection program is initiated.

7.2 Maneuvering Along the Blades

The task of flying along the individual blades was not a part of the assigned problem for this thesis. Therefore the coverage of this task will be merely anecdotal. A proposed approach follows along with potential problems and comments.

7.2.1 Choosing a target blade

The previous part of the program should have left the UAV in a position directly in front of the hub where the base of all blades are visible, similarly to figure 7.4, with knowledge about blade angles. An arbitrary blade would then be chosen, and it could be locked on by using Hough lines restricted to the corresponding angle.

7.2.2 Following the blade

When following the blade the most obvious task is to move in the direction of the blade. Since the blade should span across the whole image frame as the UAV moves along, the Hough line transform is assumed to produce a clear result making it straightforward to maintain the angle.

Another task is to keep the UAV at a suitable distance from the blade. To avoid crashing it should maintain a safe distance, but simultaneously it should fly close enough for the camera to obtain quality footage (for inspection). Approximately 5 meters is suggested. Estimating the distance using

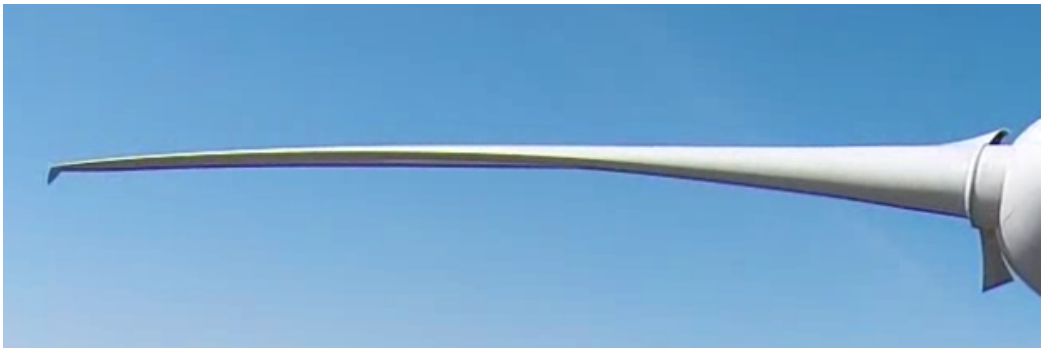


Figure 7.5: A blade of a wind turbine

the camera (via the pinhole camera model as described in section 7.3) might prove problematic because the blade varies in width from base to end (figure 7.5). Using a sonar instead, or a similar device, is presumably the better solution. The relevant distance should be suitable for accurate measurement.

Optimally the front of the UAV should point perpendicularly to the surface of the blade. If the edges of the blade were parallel, one could utilize how the angle between the edges change with perspective to determine orientation. However, the curvature of the blade makes designing such a scheme non-trivial. Perhaps a more beneficial approach would be to simply maintain a steady compass angle of the UAV which is obtainable from the IMU.

7.2.3 Navigate around the tip of the blade and back the other side

If the Hough line transform was chosen for following the blade, then the tip of the blade should be detectable by the point where the Hough lines eventually come to an end. A corner detector could also prove suitable for this task.

When moving around the tip, it is questionable how applicable the computer vision methods which have been discussed are. The problem is that the features of the wind turbine become less apparent when viewed from the side. A solution could be to instead rely on a pre-programmed maneuver like for instance a semicircle motion around the tip, as illustrated in figure 7.6.

After having arrived at the other side and again identified the blade, the UAV can return to the hub using the same blade following procedure as before, but this time from the back. When the hub is reached, the next blade can be selected for inspection.

Figure 7.6: A top-down perspective illustrating a semicircle maneuver around the tip of the blade.

7.2.4 Note on blade orientation

As can be seen for instance in figure 7.5 or 7.4, when the turbine is halted the individual blades are rotated so that their narrow side points forward, presumably to reduce the force exerted by the wind. However, this results in the wide sides pointing upward and downward, and thus the UAV which navigates the blades from the side will have a poor angle of view of these areas. Since the wide side of a blade is pointing forward (toward the wind) when the wind turbine operates, this side is more susceptible to damage. For this reason, it should be further investigated whether it is sufficient to inspect the blades from the side. If not, a more complicated navigation approach could be considered, or perhaps a more flexible camera set-up could be composed.

7.3 Distance Estiation

The navigation approach (section 7.1) relies on being able to detect when the UAV has reached a desired distance. To address this matter, some options for distance estimation will be discussed.

7.3.1 Distance estimation using video camera

The first option is to estimate distance from the camera output by using computer vision. This can be achieved by applying the pinhole camera model

(section 4.5). By rearranging equation (4.14) with respect to the object's distance to the camera, x^c, one obtains the following:

$$\begin{aligned} x^c &= \frac{f}{y^i} y^c \\ x^c &= \frac{f}{z^i} z^c \end{aligned} \tag{7.1}$$

Now, let the measured width and height of the object of interest be given by Δy^i and Δz^i. If the corresponding width and height of the same object is known in reality and given by Δy^c and Δz^c, the distance can be estimated by either of the following:

$$\begin{aligned} x^c &= \frac{f}{\Delta y^i} \Delta y^c \\ x^c &= \frac{f}{\Delta z^i} \Delta z^c \end{aligned} \tag{7.2}$$

A suitable target in the wind turbine tracking program would be the width of the tower. Since tower detection has regardless been given considerable care and has easily distinguishable edges it provides a reliable basis for this approach. One should keep in mind that the tower is usually wider at its base, therefore a specific area of the tower should be targeted. The uppermost area would possibly be the best choice because this area persists inside the field of view at all distances. Additionally, since it is desired to point the camera such that the hub is centered in the image, the area will also be close to the image center and thus not notably affected by image distortion.

This method was implemented into the program and was used for the experiments presented in section 9.2.3.

7.3.2 Distance estimation using dedicated hardware

Under appropriate conditions dedicated instruments will provide more accurate results than the camera based procedure. However, for most of the time the wind turbine will be a small target due to the distance. A sonar will for instance become inaccurate as the propagation distance increases. On the other hand, good performance can be expected at the range where the final desired distance will actually be measured. Another suggestion could therefore be to use camera for estimation at longer ranges, before relying on a sonar when getting close.

7.4 Yaw Estimation

It is desirable to approach the wind turbine from its direct front. This will both ascertain the position of the UAV in relation to the wind turbine and ensure the approach being made from a safe angle. For this purpose, a

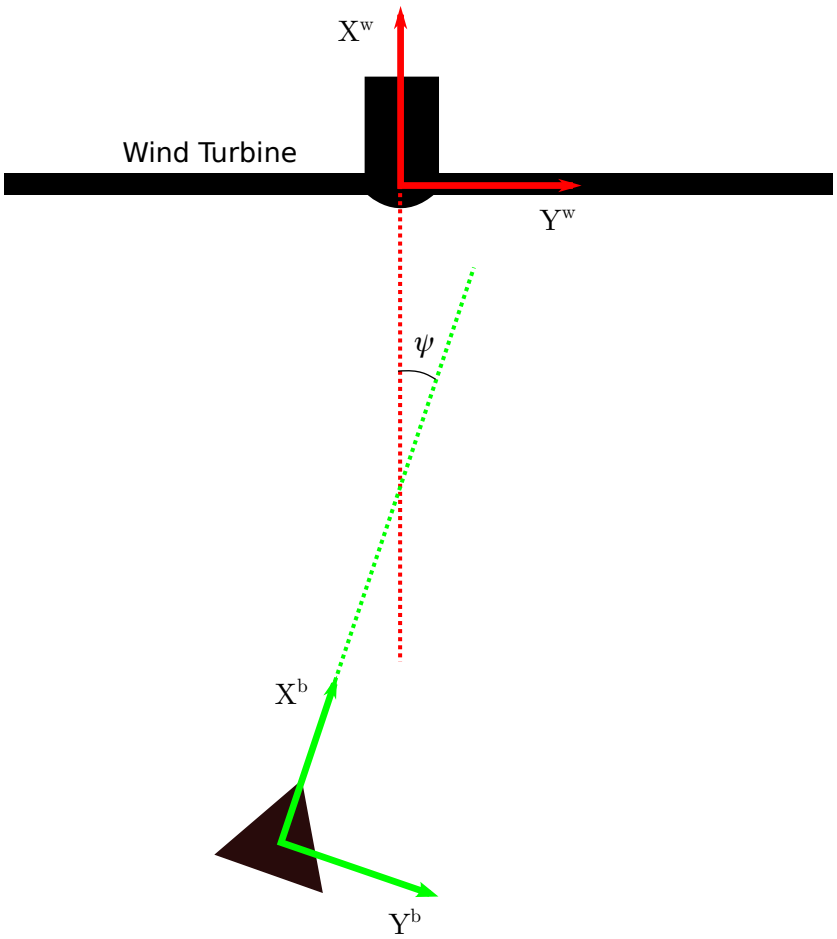


Figure 7.7: An illustration of the yaw orientation between the b-frame and the w-frame.

method is sought to estimate the yaw orientation between the UAV and the wind turbine i.e. the yaw component of R_w^i (figure 7.7). As seen from the figure, the coordinate frames are defined in such a way that when the UAV is directly in front of the hub and facing the wind turbine, the X axes are aligned and $\psi = 0$.

7.4.1 Estimation via coordinate transformation

One way to approach this issue is to examine how an yaw rotation relates to the image frame. For simplicity it is assumed that the z-axes are parallel (i.e. no roll or pitch rotation), which is fulfilled if the UAV hovers stably. If so, a certain vector, $\mathbf{v}^w = \begin{bmatrix} x^w & y^w & z^w \end{bmatrix}^T$ in the wind turbine frame can be rotated to the camera frame (which is essentially identical to the body frame) as follows:

$$\mathbf{v}^c = \mathbf{R}_w^c(\psi)\mathbf{v}^w \tag{7.3}$$

where the rotation matrix is a pure yaw rotation.

$$\mathbf{R}_w^c(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.4}$$

In order to transfer points ($\mathbf{p}^c = \mathbf{T}_w^c\mathbf{p}^w$) the position also needs to be accounted for and transferred. Before doing that, another simplification will be made. It will be assumed that the hub of the wind turbine is centered in the image frame, so that the x^c-axis intersects the origin of the wind turbine frame. In practice, this is achieved by through T1 in the navigation approach (7.1). The consequence of this simplification is that the distance between the origins of the two coordinate frames, denoted $\mathbf{r}_{cw}$, can be expressed as a length purely along the X^c axis. Expressed in the camera frame this becomes the following:

$$\mathbf{r}_{cw}^c = \begin{bmatrix} x_{cw}^c \\ 0 \\ 0 \end{bmatrix} \tag{7.5}$$

Points can then be transferred by using the homogeneous transformation matrix, which is defined as:

$$\mathbf{T}_w^c = \begin{bmatrix} \mathbf{R}_w^c & r_{cw}^c \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{7.6}$$

The method also requires expanding the dimensionality of the point such that: $\mathbf{p}^c = \begin{bmatrix} p_x^c & p_y^c & p_z^c & 1 \end{bmatrix}^T$, $\mathbf{p}^w = \begin{bmatrix} p_x^w & p_y^w & p_z^w & 1 \end{bmatrix}^T$

Thus one obtains

$$\mathbf{p}^c = \mathbf{T}_w^c \mathbf{p}^w = \begin{bmatrix} \cos\psi & -\sin\psi & 0 & x_{cw}^c \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \\ 1 \end{bmatrix} \tag{7.7}$$

$$\begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \\ 1 \end{bmatrix} = \begin{bmatrix} p_x^w \cos\psi - p_y^w \sin\psi + x_{cw}^c \\ p_y^w \cos\psi + p_x^w \sin\psi \\ p_z^w \\ 1 \end{bmatrix} \tag{7.8}$$

Using the pinhole model, equation (4.14) from section 4.5, the point can be expressed in the image plane as:

$$\begin{bmatrix} p_y^i \\ p_z^i \end{bmatrix} = \frac{f}{p_x^w \cos\psi - p_y^w \sin\psi + x_{cw}^c} \begin{bmatrix} p_y^w \cos\psi + p_x^w \sin\psi \\ p_z^w \end{bmatrix} \tag{7.9}$$

By looking at how the wind turbine frame is defined, as for instance illustrated in figure 7.7, one can observe that the relevant points of the wind turbine lie close to the $Y^w - Z^w$ plane , such that $p_x^w \approx 0$. By using this approximation the equation is greatly simplified.

$$\begin{bmatrix} p_y^i \\ p_z^i \end{bmatrix} = \frac{f}{-p_y^w \sin\psi + x_{cw}^c} \begin{bmatrix} p_y^w \cos\psi \\ p_z^w \end{bmatrix} \tag{7.10}$$

Using the lower low yields the most practical expression:

$$p_z^i (p_y^w \sin\psi - x_{cw}^c) = -f p_z^w \tag{7.11}$$

$$\psi = \arcsin\left(\frac{\frac{f p_z^w}{p_z^i} - x_{cw}^c}{p_y^w} \right) \tag{7.12}$$

7.4.2 A more problem specific yaw estimation approach

Rather than using a general approach as described above, a better option could be to use a more specialized approach which takes advantage of the features of the wind turbine. A method will now be presented which utilizes the depth variation of the wind turbine.

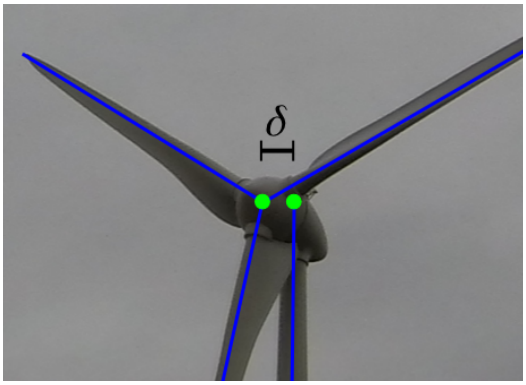


Figure 7.8: Illustrates the gap between the hub center and tower which appears as the yaw angle increases.

One can observe from figure 7.8 that as the yaw angle increases, two points which were initially behind each other drift apart. In particular, the center of the hub and the tower of the wind turbine, which are among the easiest points to determine, form the gap denoted δ.

δ, which is a distance given in the image frame, can be expressed as a function of the actual distance between the two points in 3D ($|p_1^w - p_2^w|$) along with the yaw orientation (ψ) and the distance from the camera (d). Using the pinhole camera model:

$$\delta = \frac{f}{d}|p_1^w - p_2^w| \sin \psi \tag{7.13}$$

Which yields the following relation

$$\psi = \arcsin\left(\frac{\delta d}{f\|p_1^w - p_2^w|}\right) \tag{7.14}$$

This method was implemented into the program and was used for the experiments presented in section 9.2.4.

7.4.3 Compass Approach

If the yaw orientation of the wind turbine relative to the NED frame is known, it could be compared to the measurements of the on-board compass of the UAV. However, the orientation of the wind turbine changes in correspondence with the wind direction to optimize efficiency. So either the wind turbine would have to be stopped in a predetermined orientation before shutting down for inspection, or the orientation has to be measured and updated to an interface accessible to the UAV.

Chapter 8

Designing the Program

Chapters 6 and 7 described the approaches which are used in the program. In this chapter, it will be explained how these parts are implemented into a complete application.

8.1 The Program Flow

The program runs in a loop where a new iteration is initiated at the arrival of a new image frame. How the individual iteration is structured will now be explained in a step-by-step fashion. A corresponding activity diagram of the program flow is given in figure 8.1. Under way, it will also be referred to some of the main functions and data structures which are used, for which a class diagram is shown in figure 8.2.

Obtaining an image frame

The first task is to obtain an image frame. Every iteration in the computer vision task processes one separate frame. The image may come from either an actual still image, a frame from a video sequence or a frame from a camera stream. Only the camera stream is relevant for the final system in actual operation, whereas image and video input is used for development.

Using an image as input simply involves reading the file. Since videos are intended for off-board testing there are no time restrictions, therefore the next frame in the sequence is simply read when the previous iteration is finished.

For the camera stream, however, it is important that the program is able to keep up. If not, one may experience the program working with frames which were actually produced a while earlier. Naturally, this could lead to disastrous results if employed in flight. To prevent this issue, a thread

is created which runs in parallel to the main computer vision task. This thread continuously captures the stream from the camera and pushes the frames communication bus, thus preventing the frame queue from stacking up. When the main task is ready to receive a new frame, the most recent frame is readily available on the bus.

A received frame is stored in the *Image* class which contains the original image along with all derived images from processing methods and images for the GUI which provides visual feedback.

Downsampling

Because the image resolution has a negative effect on the efficiency of most image processing methods, it is desirable to limit it to a reasonable level. This is done in two different ways. The first is to directly instruct the camera to stream at a lower resolution. In this case, the downsampling step is skipped, and no processing is needed locally by the computer thus reducing computational demand.

A second approach was implemented in case either the camera does not support to limit its resolution or if a video or image is used as input. In this case the image is down-sampled using the Gaussian pyramid (section 4.6). It is done using the function *downsampling()*, from the *Image* class.

The image resolution should lie in the range of 200-500 pixels for both axes. This offers an acceptable trade-off between performance and image quality. In addition, keeping the resolution in a specified range yields consistent results from processing methods which are scale dependent. For instance, most edge detectors work by comparing the current pixel to the nearest neighbouring pixels. Thus, when the image resolution is increased, the same amount of neighbouring pixels cover a smaller portion of the whole image, leading to decreased sensitivity for the method.

Gaussian blur

In some cases it is desirable blur the image before proceeding to the Canny edge detector and Hough transform. The reason is that if the image is sharp, the edge map from the Canny detector can become very dense and noisy. Since the Gaussian blur is in practice a low pass filter, these effects would be reduced and a more predictable result is obtained. Care should however be taken, as excessive blurring can eliminate wanted the features in the image.

Using the camera configurations in this work, however, blurring needed not to be done directly. It was done indirectly via downsampling when using the high resolution GoPro videos. Also, the video stream from the USB

camera, which was configured to readily deliver low resolution images, did not deliver notably sharp images.

Canny edge detection

To provide the binary edge map required by the Hough line transform, the Canny edge detector (section 4.3.1) needs to be executed first. It is available from the OpenCV library as *Canny()*. The high and low hysteresis thresholds in the detector should be set relatively high (thus decreasing sensitivity), since the edges produced by the contours of the wind turbine are generally rather prominent compared to most edges exhibited by features in the environment. By consequence fewer irrelevant edges are included, thus reducing computation time for the detector itself and for the subsequent Hough transform which will detect fewer lines.

The resulting edge map is saved in *Image.canny*

Hough line transform

Having obtained the edge map from the edge detector the Hough line transform can be executed. This is done by *generateHoughLines()* which utilizes the function *HoughLinesP()* from OpenCV, and stores it in the class *HoughTransform* as *HoughLine* objects.

Recognition

Next the recognition algorithm is performed as descrived in section 6.1.5. *findTowerLines* and *findBladeLines* detects and labels Hough lines which matches the conditions. A *WindTurbine* object is created, where the lines are grouped into *Blade* and *Tower* objects. The remaining logical operations are performed using *findBladeAngles()*, *findTowerAngle()*, *removeFalseBlades()*, and *findHubCenter()* from the *WindTurbine* class, and hopefully a successful detection is obtained with an estimated hub center position.

The rest of the relevant functions in the *WindTurbine* class are performed to perform all logical operations required to finally obtain a successful detection along with a position estimation.

If the detection failed, i.e. the tower was not detected or insufficient amount of blades were detected, then the program **gets prediction from Kalman filter** as the current position estimation, by updating the Kalman filter and extracting *HubCenterKF.statePt*. But if this has already happened for a prolonged period of time, movement should be halted a **search procedure** should be initiated. This could include simple yaw rotation until the wind turbine is spotted, and if also this should fail the best course of action

might be to let GPS navigation to take over and return to the initial start position.

Update Kalman filter

If detection was successful, the estimated position is added as a measurement to the Kalman filter. The filter is updated using *HubCenterKF.update()* and filtered position obtained from *HubCenterKF.statePt*.

Estimate yaw and distance

Next, the yaw angle and distance are estimated using *estimateWindTurbineYaw* and *estimateDistance* from the *WindTurbine* class, using the approaches described in section 7.3 and 7.4.

Maneuver towards the wind turbine

Finally, based on estimated data, the wind turbine sends the commands in correspondence with the approach described in section 7.1, and goes back to start to receive a new image frame.

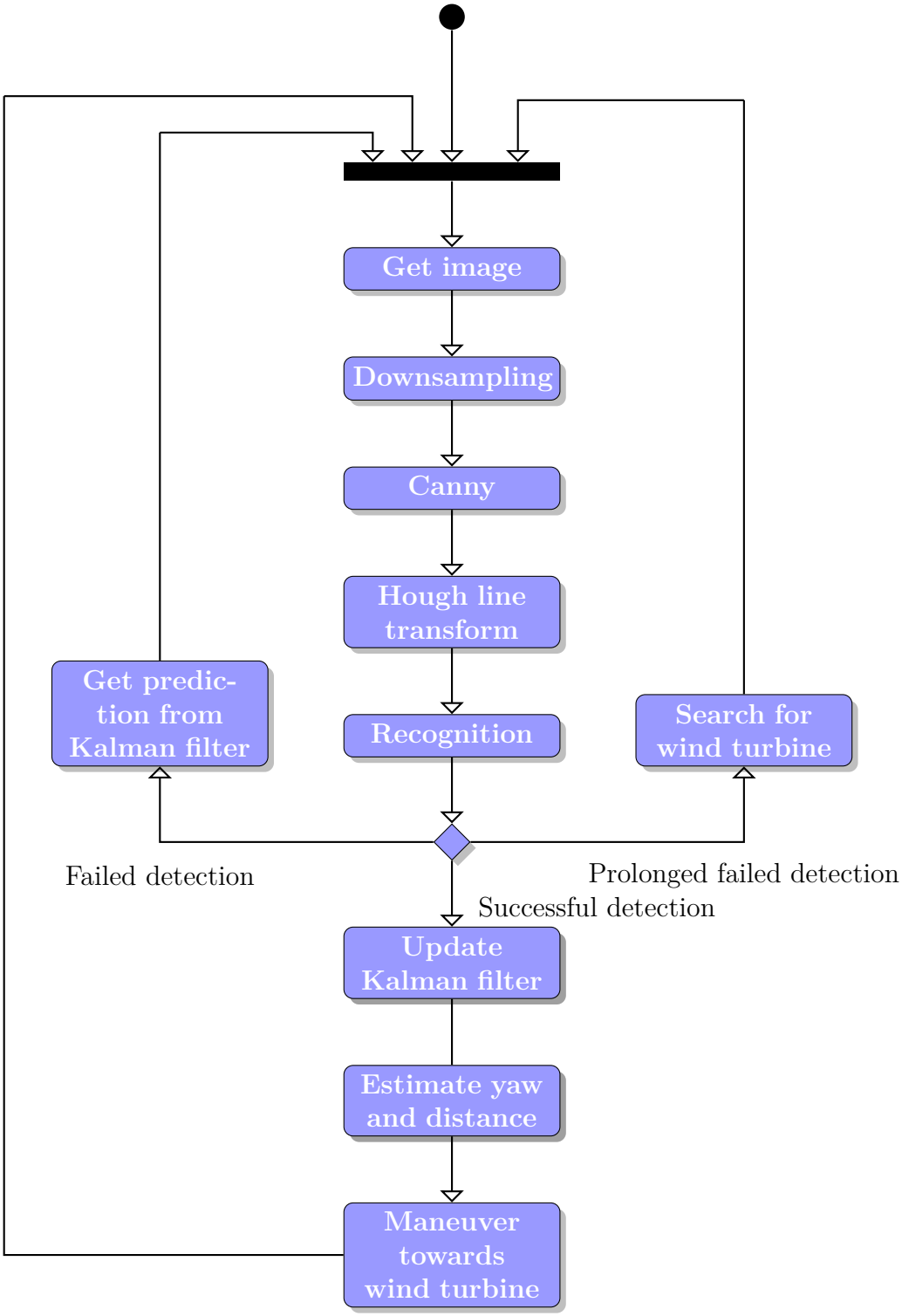


Figure 8.1: Activity diagram describing the flow of the computer vision program.

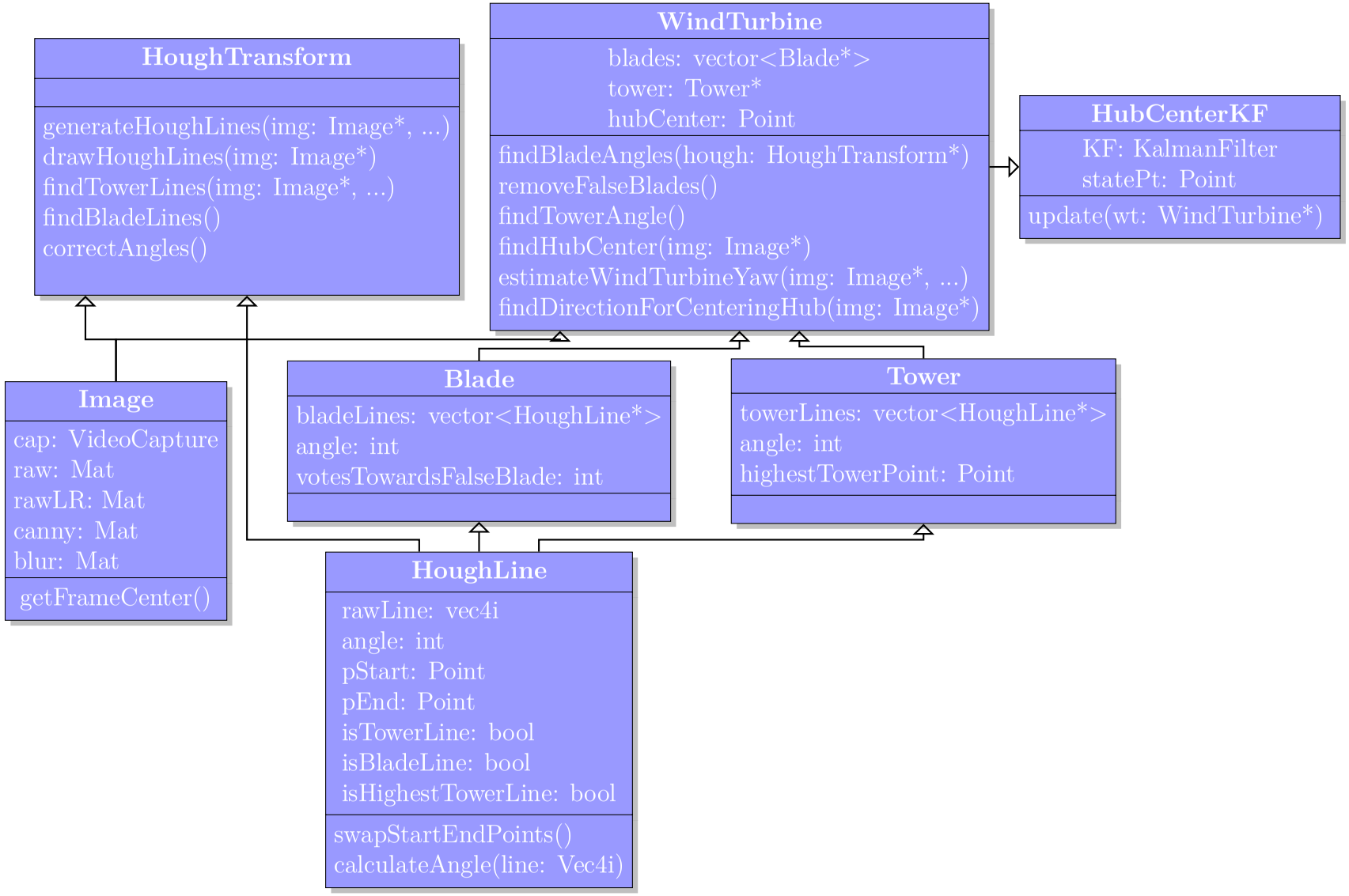


Figure 8.2: Class diagram of the computer vision program.

Chapter 9

Results

This chapter will examine the performance of the system through a variety of experiments. Several aspects of the program were investigated to get an overall impression. First, an overview is given over factors which affect whether a detection is successful and its accuracy. Next, the program is run using different video sequences as input, and the performance for estimates of hub position, distance and yaw angle is evaluated. Lastly, the computation requirements were investigated, and an on-board test was performed to observe what iteration frequencies are feasible for maintaining stable control.

9.1 Factors which affect Recognition Performance

The aim of this section is to evaluate what conditions affect the recognition algorithm, in order to get an overall understanding of what sources of error are present. To do this, it will be attempted to isolate the circumstances which disturb or disrupt a detection, accompanied with representative images.

In this analysis, the terms successful or failed detection will be used. This simply indicates whether a wind turbine was detected in the image. It does not, however, imply anything about the accuracy of estimated position, angle or distance.

9.1.1 Minimum requirements for successful detection

In order to get a successful detection there certain conditions need to be fulfilled. These can be divided in two parts. The position of the wind turbine and tuning matching environment conditions.

The restrictions of the position of the wind turbine are posed directly and indirectly by the program itself. First, the tower of the wind turbine has to be in view, since the recognition algorithm begins by searching for tower lines. Since the only angles sufficiently close to 90 degrees are accepted, it is required that if the UAV currently has a significant roll angle the IMU must provide a good enough estimation to compensate. Furthermore the tower must reach the lowest 20% of the image. In addition, at least two blades have to be in view such that an intersection can be calculated to estimate the hub center. The yaw orientation must also be sufficiently small so that the angles between the blades are not skewed past the range of 100-140 degrees.

The second set of conditions are related to the image processing parameters. If the image properties vary too much from what the parameters are tuned against, the output from the Canny edge detector or the Hough Transform may not be suitable for further analysis. Image properties can differ if the camera is changed, however should not be a problem if a single camera is used in practice. More important are lighting and weather conditions, which alter the suitable range for the parameters. This will now be examined.

Parameter dependency

The Canny edge detector is the critical part under lighting variation. A change in lighting can affects the intensity distribution in the image, and can thus alter the strength of the edges. The hysteresis threshold values need to be adjusted low enough to accept the most prominent edges, but high enough to filter weaker unwanted edges (section 4.3).

A brief test was run on a single video clip to examine the threshold ranges which produced successful or meaningful detection. First the threshold values were progressively increased. It was found that the edges of the wind turbine began to diminish at about the following values:
High hysteresis threshold: 500
Low hysteresis threshold: 400
Next the values were lowered. The amount of noise was having a significant impact on accuracy of detection when the values reached the following:
High hysteresis threshold: 200
Low hysteresis threshold: 100

It should be emphasized that the transitions from good to bad were very gradual, and these limits are thus fairly subjective. In addition, there is an interplay between the high and low threshold which is hard to grasp. Nevertheless, the point persists that the range at which the method produces useful output is not too narrow to allow room for environment change or suboptimal tuning.

Table 9.1: Chosen parameters for the Hough line transform

Hough threshold	43
Minimum line length	50 pixels
Maximum gap in lines	13 pixels

The Hough line transform, on the other hand, is a different matter. Because the shape of the wind turbine always remains the same, a specific set of parameters will produce similar results unless the Canny detector outputs a drastically different edge map, in which case the Canny detector should be adjusted. By experimentation the parameters in 9.1 were found to work well under any tested lighting conditions. It should be noted that since some of the parameters are given in pixels, they would have to be altered if a significantly different display resolution were to be used.

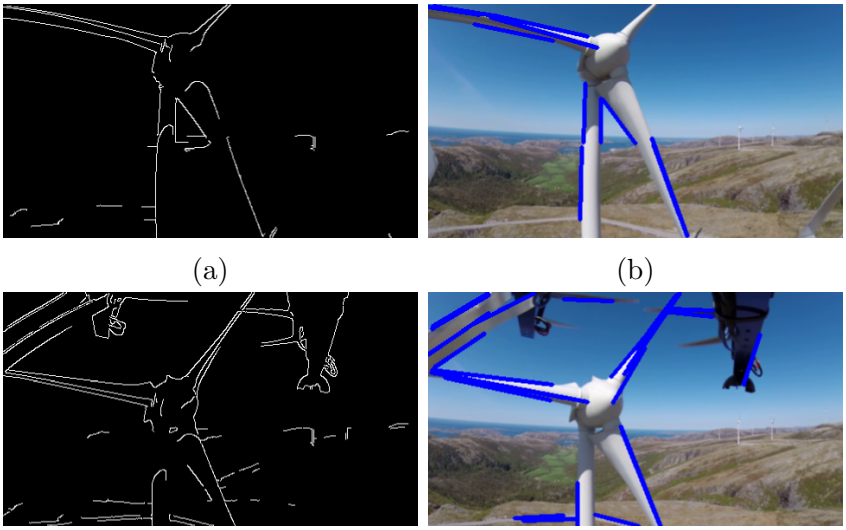
9.1.2 Failed detections

Assuming the minimum requirements for successful detection are met, it will now be investigated how else a detection can fail. A failed detection is simply when a wind turbine is not recognized anywhere in the image. This can occur when either the tower was not detected, or in case the tower was detected the sufficient amount of valid blades was not detected. These cases will now be examined.

Failure to detect tower

In order to detect a tower, one merely needs to identify a sufficiently long vertical line starting inside the lower 20% of the image. The length requirement fails if the wind turbine appears too low in the image, such that just a small part of the tower is visible. If the minimum length for Hough lines is set to 50 pixels (with image height of 270 pixels), as it was in these experiments, the hub cannot fall below the lowest fifth of the image height.

If there is an interference such that the tower edges are split into smaller lines, it can lead to failed detections at higher hub positions. Such an issue arose with the appearance of a shadow cast by the hub onto the tower. Figure 9.1 shows how this shadow disrupts the canny detector and consequently also the Hough transform. Worse than segmenting the lines, it also can hide the edges completely in the shadow area if the background is of matching color, such that the highest point among the tower lines (which form the center for the blade search area) is lower than it should be. The effect is stronger when



(a) (b)

(c) (d)

Figure 9.1: (a) Shows how the shadow under the hub causes a split left edge of the tower which in (b) results in two separate Hough lines. (c) Shows how the same shadow blends into the background and hides the edge, which leads to no Hough line in (d).

the distance to the wind turbine is shorter, because then the shadow appears larger in the image.

Failure to detect blades

Since the estimated top of tower was based on the highest point among the tower lines, this point may in some cases be found too low, as in figure 9.2b. Again, the shadow is the cause of the problem. Blades are thus not detected, since they have to be close to the estimated top of tower position which defines the center of the blade search radius. Another scenario is when too many (incorrect) blades are detected, either due to a poor Hough transform or too many disturbances in the image. Then there is much uncertainty as to which detections are true blades, and the voting procedure (described in section 6.1.5.2) will discard all detections.

9.1.3 False positives

A successful detection does not necessarily imply a correct detection, due to the occurrence of false positives. In this context, false positives refer to detected tower lines or blade lines which do not actually correspond to the

(a) (b)

Figure 9.2: (a): Failed detection because the tower does not exhibit any lines which are long enough to be detected. (b): Failed detection because the highest point among the tower lines (blue circle) is not high enough and thus the blade search radius (green) does not capture any of the blades.

true edges of the tower and the blades.

A false tower detection was only experienced when another wind turbine was visible in the image, close enough to produce a long enough line, and at a specific height such that the bottom of the tower was in the lower 20 % of the image. This had no effect on successful detection, because the blade search radius is centered at the highest point among the tower lines, and the correct tower would always reach higher.

On the other hand, false blade detections occurred more frequently. Every detected line which has an end point inside the blade search radius yields a true or false positive. This was most commonly caused by the horizon and the visible parts of the hexacopter itself, and to a lesser degree the environment when the blade search radius was big (at close ranges). However, the environment produced very few edges in general. Many of the false edges are however removed by the voting procedure (section 6.1.5.2) or are dominated by the usually more numerous true blade edges.

9.2 Runtime performance using video input

In this section the program will be evaluated by using video sequences as input. The videos were recorded by at Bessakerfjellet wind farm, Norway (appendix B.2). For recording a GoPro (appendix C.1) camera was used. The camera was mounted on the same type of hexacopter as described for the system, but with no payload except for the bare minimum required for flying via remote control. A gimbal was also used to stabilize the camera. The recordings were done in such a way to attempt to mimic the maneuvering plan

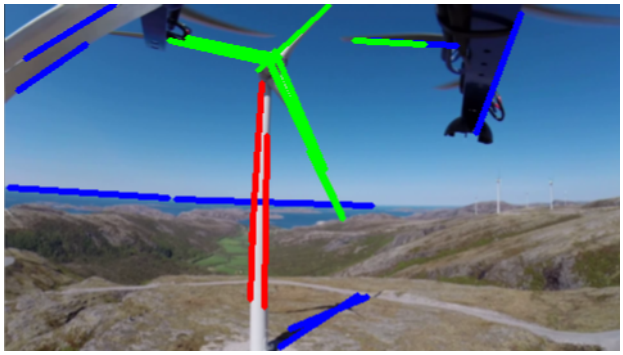


Figure 9.3: The interference of the hexacopter is an example of where the edge of another entity is falsely detected as a blade edge, illustrated by the incorrect green line.

(described in 7.1) for an inspection scenario, in order to get representative footage at the expected range and angles.

When evaluating the performance, the estimated hub center position from the program will be compared to the actual hub center position. The actual position was measured by following the position in the video manually using a computer mouse. For this reason the measured actual position can be off by a few pixels and lag slightly behind under quick movements, but still provides useful means for comparison.

The video sequences are downscaled from the original GoPro display resolution to 480x270. In the plots, the pixel positions are bounded to match these numbers, such that the edges of the plot reflect the edges if the image. Furthermore, since the pixel coordinates count y-position from top to bottom, the y-axis of the plots is inverted when representing the y-pixel position in order to match the direction visually.

In addition, the detection values will be accompanied by a Boolean plot which indicates whether the detection was successful (green) or failed (red) for each frame. If a detection is false for a frame and no Kalman filter have yet been added, the output value will simply be equal to the value from the last successful detection.

Three of the video clips will be presented, each captured at a different distance. Because the hexacopter was remotely controlled from the ground and was flying at about 50 meter height, exact distances between the hexacopter and the wind turbine were not obtained. The distances were however estimated during flight to be around 30 meters, 15 meters and 8 meters for the first, second and third clip respectively.

In section 6.1.5.2 it was mentioned how the search radius can be adjusted

Table 9.2: The search radius used for each distance. Radii are given as fraction of image height

Clip #	Distance [m]	Search Radius
1	25	0.3
2	15	0.5
3	8	0.9

(a) (b) (c)

Figure 9.4: Representative frames for the three video clips, and also showing the size of the blade search radius used for each clip. (a): Video clip # 1. (b): Video clip # 2. (c): Video clip # 3.

to according to distance. Using this approach the search radii given in table were 9.2 found and used. How this relates to each video clips can be seen in figure 9.4. The figure also provides a representative image of how the wind turbine appears from the UAV's point of view from the given distance.

9.2.1 Position estimation

The Canny edge detector was set to the same bounds for all the three clips, which provided a good balance between desired detection and noise rejection:
High hysteresis threshold: 400
Low hysteresis threshold: 300

Long distance, video clip # 1

The result described here can be viewed in the file "Clip 1 estimation.mp4" in the digital attachment.

First, the performance was examined for a video clip where the distance was long, about 30 meters (near the initial position of the maneuvering plan). A representative frame from the clip is shown in figure 9.4a. During the clip the wind turbine moves from the top to the bottom of the frame while distance remains roughly constant. Figure 9.5 show estimated hub center

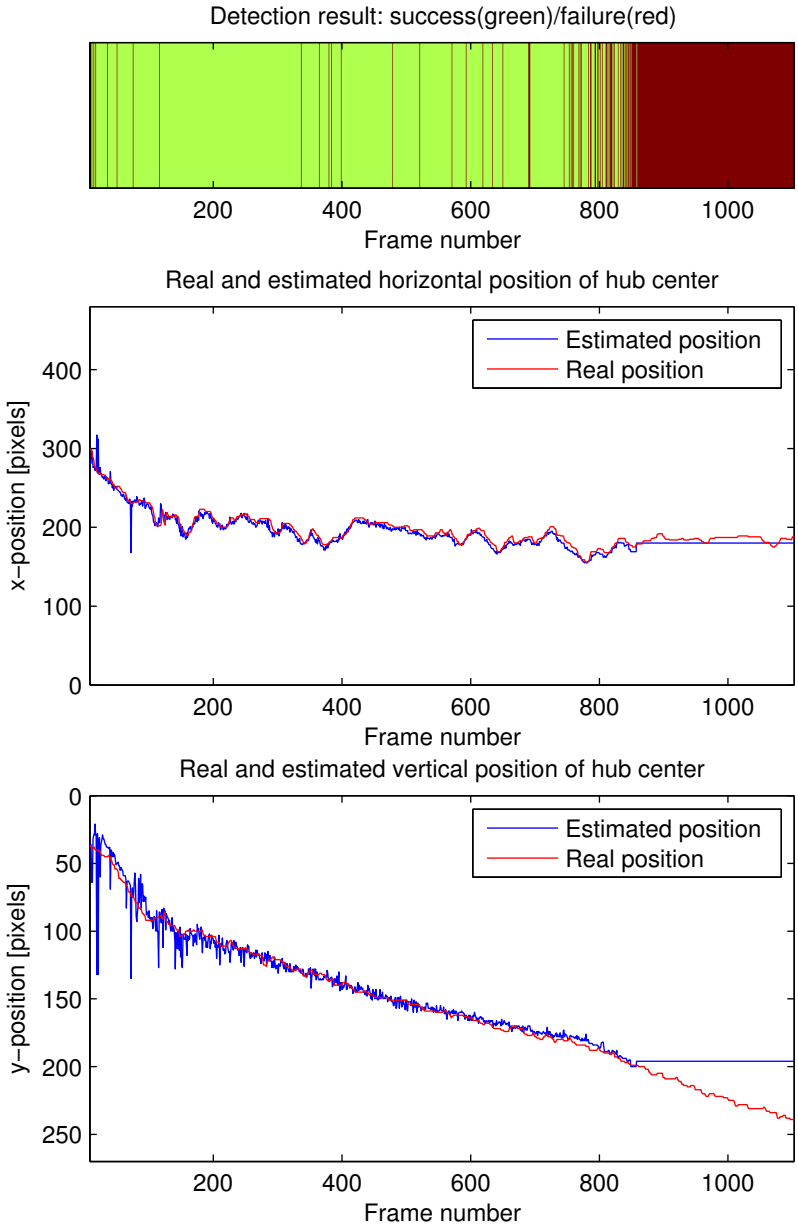


Figure 9.5: Video clip # 1 detection results where the distance from the wind turbine is around 30 m. The estimated position follows the real location reasonably well. Detection is lost when y-position approaches the lower edge.

positions, both horizontal and vertical, along with the Boolean plot showing success/failure.

The results shows accurate tracking for this range. In x-position the tracking varies slightly less than in y-position where there are some occasional spikes in the y-position towards the center caused due to false positives by interference from the horizon. It is observed, as expected, that the program loses detection when the wind turbine reaches a sufficient low y-position around the lower third of the image. This happens due to failed tower detection, since the remaining visible parts of the tower produce too short Hough lines. The shadow located under the hub is also shortening the Hough lines, such that detection is lost a bit earlier than it would under optimal conditions (just as was explained in section 9.1.2).

Medium distance, video clip # 2

Next, the program was examined for a clip recorded at a closer distance, around 15 meters, as represented by figure 9.4b. The results shown in figure 9.6 were obtained. Position estimation is still rather accurate, but compared to the results from video clip # 1, there is both slightly more noise in the estimations along with a higher number of detection failures. The increased noise can be seen as a consequence of the expanded search radius. When a larger area is covered, the amount of false positives is also potentially increased. The new interfering entities were primarily the propellers from the hexacopter itself along with the most distinct contours of the landscape. The amount of false positives (falsely detected as blades) is also responsible for an increased amount of false detections, because the actual blades cannot be distinguished from the false detections.

The shadow on the upper part of the tower does now interfere more than in video clip 1, because due to the closer distance it appears bigger in the image. This causes the detection to be lost just before the y-position reaches the lower half.

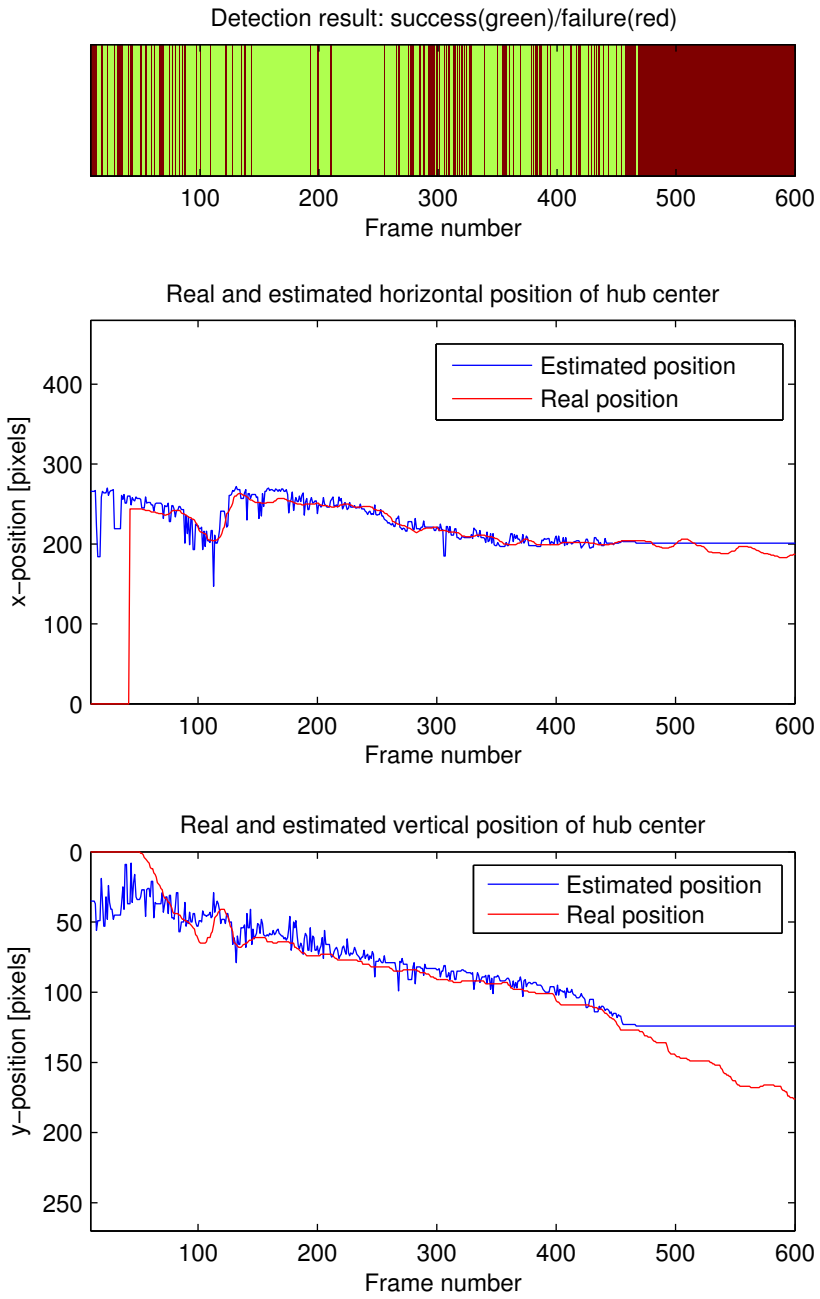


Figure 9.6: Video clip # 2 detection results where the distance from the wind turbine is around 15 m. The estimated position follows the real location reasonably well. Detection is lost when y-position approaches the lower edge, but earlier than at the longest range.

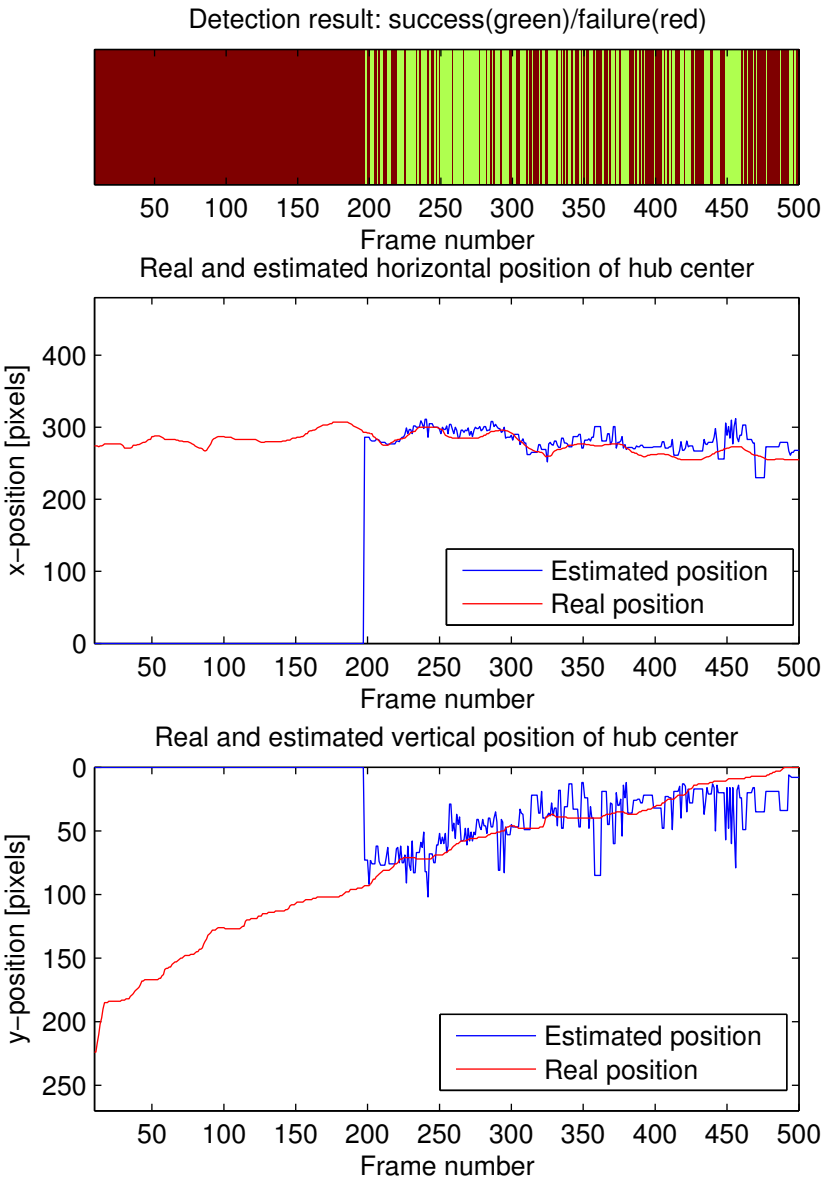


Figure 9.7: Video clip # 3 detection results where the distance from the wind turbine is around 8 m. Successful detection is obtained only at the upper third of the image. Detection is also less reliable.

Short distance, video clip # 3

The results from video clip 3 (a representative image shown in figure 9.4c) are shown in figure 9.7. This was the nearest recording, filmed at around 8 meters from the wind turbine. The issues which arose at medium distance, were even more prevalent here. Successful detections were not achieved before the wind turbine had reached the uppermost third of the image, due to the disrupting shadow covering an even larger area.

Since the blade search radius was set bigger than at medium distance, even more false positive blade detections were present, causing inaccurate position estimation and a significant amount of failed detections. In addition, towards the end the upper blade disappears from view, and thus other falsely detected detected blade lines are more easily accepted. Especially the propellers caused much interference and are mainly responsible for the bias in y-position towards the end.

9.2.2 Position estimation with Kalman filter

The program was then again run using the same video clips as input and using the exact same configurations, but with Kalman filter tracking enabled as described in section 6.2. The $\mathbf{Q}$ and $\mathbf{R}$ matrices were experimentally tuned to the following
$\mathbf{Q} = 10^{-4} \cdot I_{2x2}$
$\mathbf{R} = 10^{-1} \cdot I_{2x2}$

This configuration assumes weak covariance in process noise, i.e. movement of the hub center in the image. On the other hand, heavier covariance in measurement noise is assumed, which achieves valuable noise reduction of inaccurate estimations. This matches the observation that, especially at short distance, the measurement error varies significantly more than the actual position.

For video clip 1, which initially had rather good estimation results (figure 9.5), a slight improvement can be observed after filtering, see figure 9.8. The spikes in y-position towards the center caused by interference by the horizon are practically removed, and a smoother tracking is achieved in overall. See direct comparison in y-position in figure 9.9.

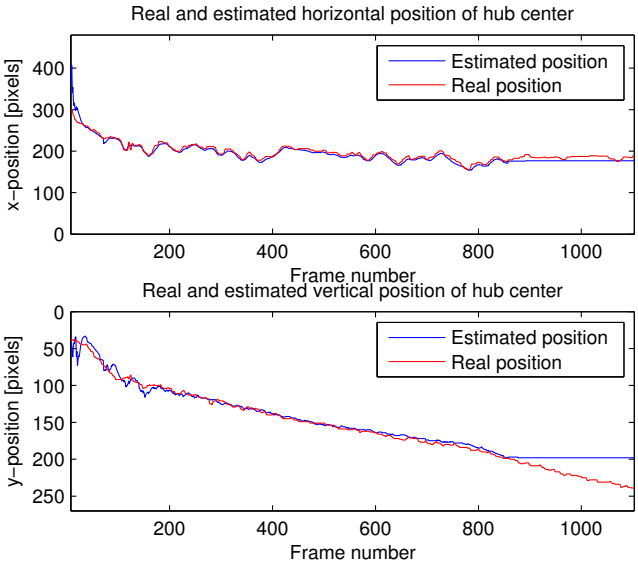


Figure 9.8: Video clip # 1 tracked position using Kalman filter. Shows a smoother result than without the filter (figure 9.5).

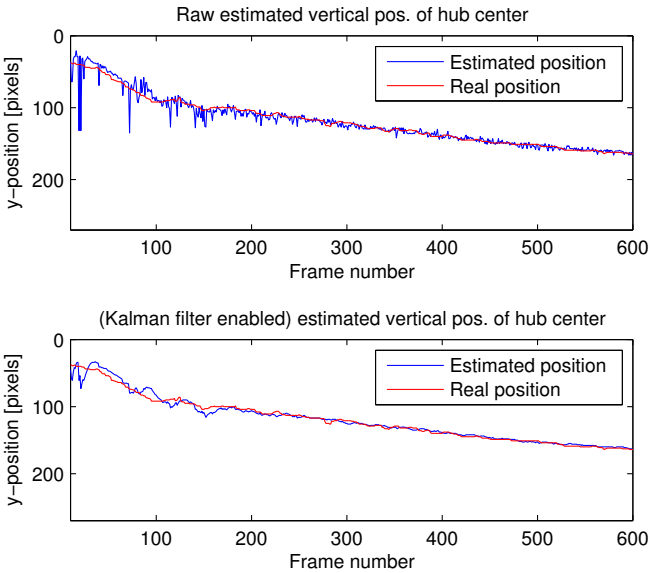


Figure 9.9: A copmarison between Kalman filtered and raw y-position estimation. Video clip # 1, was used as input.

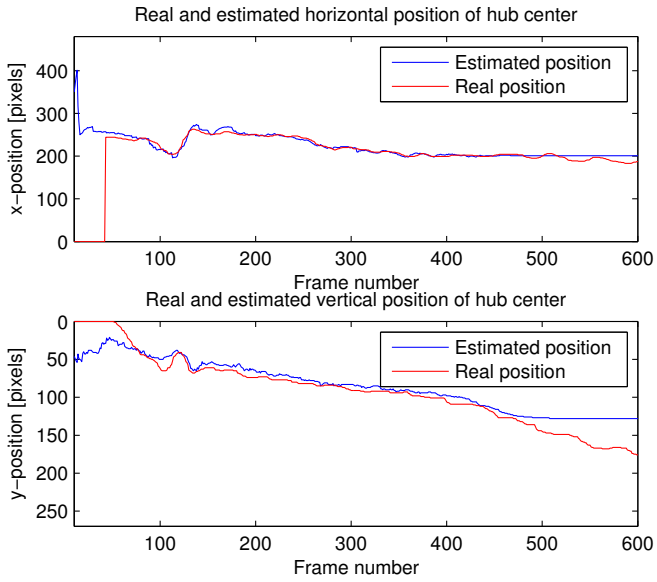


Figure 9.10: Video clip # 2 tracked position using Kalman filter. Shows a smoother result than without the filter (figure 9.6).

A similar observation is made for the middle distance video clip 2. The filtered tracking results, shown in figure 9.10, show a favourable decrease in noise compared to the unfiltered detection (figure 9.6) which had a significant amount of false positive detections present.

The filtered tracking results from video clip 3 are shown in figure 9.11. The noise from the unfiltered date (figure 9.7) is enough to produce significant deviation from the real position, especially in the y-direction. The bias in y-position towards the end could also not be corrected. This differs from what was seen for video clip 1, where spikes towards the middle were essentially removed since there were enough accurate estimations in between.

Arguably this could still be used to determine control commands, during the parts of successful detection. The prolonged sequences of failed detection can, however, not be remedied by the Kalman filter.

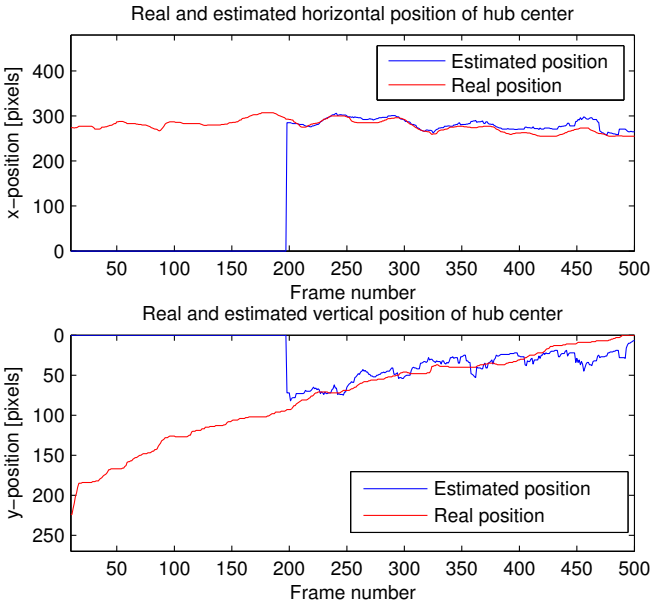


Figure 9.11: Video clip # 3 tracked position using Kalman filter. Shows a great improvement over the unfiltered result (figure 9.7).

9.2.3 Distance estimation

Distance was estimated using equation (7.2) from section 7.3. The width of the tower was used as target for estimation. This was done by first finding the outermost tower lines, and extending them to a common height and measuring the length between them (in pixels). It was considered to use the highest point among all tower lines as the common height, but since this point can vary significantly between frames the bottom edge of the image was chosen instead. This causes a more stable estimation, but will correspond to a wider point of the tower.

When using the same Canny detector parameters as for position estimation, a shadow along the right edge of the tower caused the edge often to be detected closer to the center than it should be or to not be detected at all. For this reason, the Canny edge detector bounds were lowered to the following:
High hysteresis threshold: 300
Low hysteresis threshold: 150

Next, the estimated focal length value of the GoPro camera was experimentally estimated to be approximately f = 0.33. Furthermore, the actual width of the tower was assumed to be 2 meters at the top, but one needs to keep in mind that the width increases towards the bottom.

The results are shown in figure 9.12. Video clip # 3 was excluded even with the lowered canny thresholds the right tower edge would not be detected. It is observed that the resolution on the distance estimations is limited, which is due to the small display resolution in the image were the tower width is measured. For instance, a single pixel in difference causes a jump in distance estimation from 35.2 m to 39.6 m.

As mentioned earlier, the exact actual distances were not measured, but were estimated to lie around 30 m and 15 m for video clip # 1 and # 2 respectively. The results show estimations lie a bit further than this in general. For clip # 1 and towards the end of clip # 2, the actual distance is roughly unchanged, but the results showed the estimated distance increasing. This is because the hexacopter was flying upwards, and thus a higher part of the tower (which is thinner) is measured. The spikes almost reaching zero from video clip # 1 are caused by detection of a tower of another wind turbine. In video clip # 2 there is much variation at the beginning towards the 150th frame because the hexacopter moves fast forwards and upwards.

It is clear that to get a more precise distance estimation, a more distinct length needs to be measured in the image.

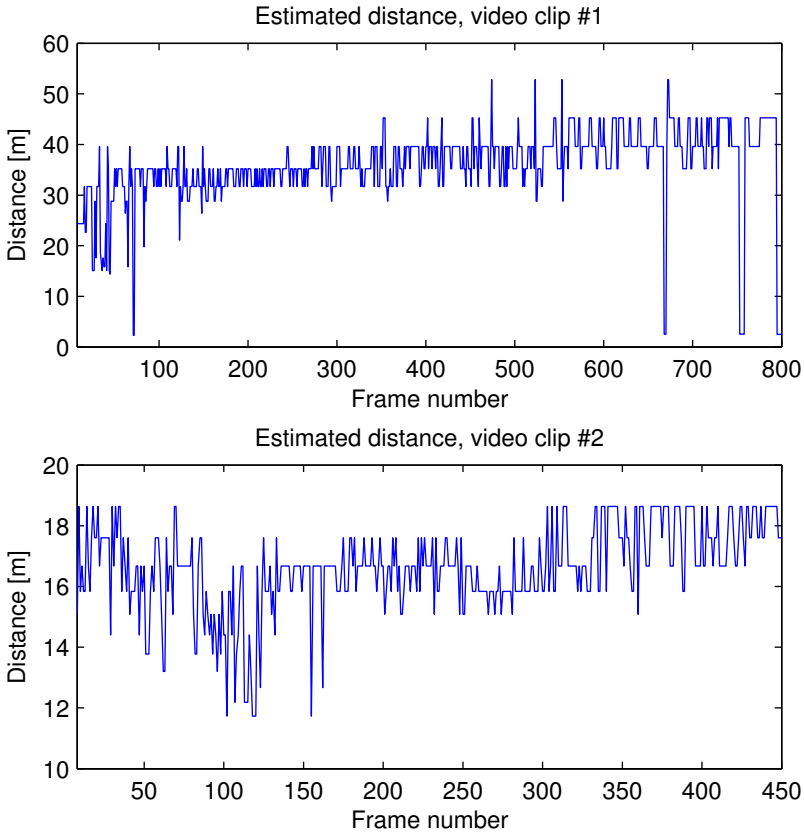


Figure 9.12: Estimated distances for video clip # 1 and 2. Actual distances were roughly estimated to lie around 30 and 15 meters for clip # 1 and 2 respectively.

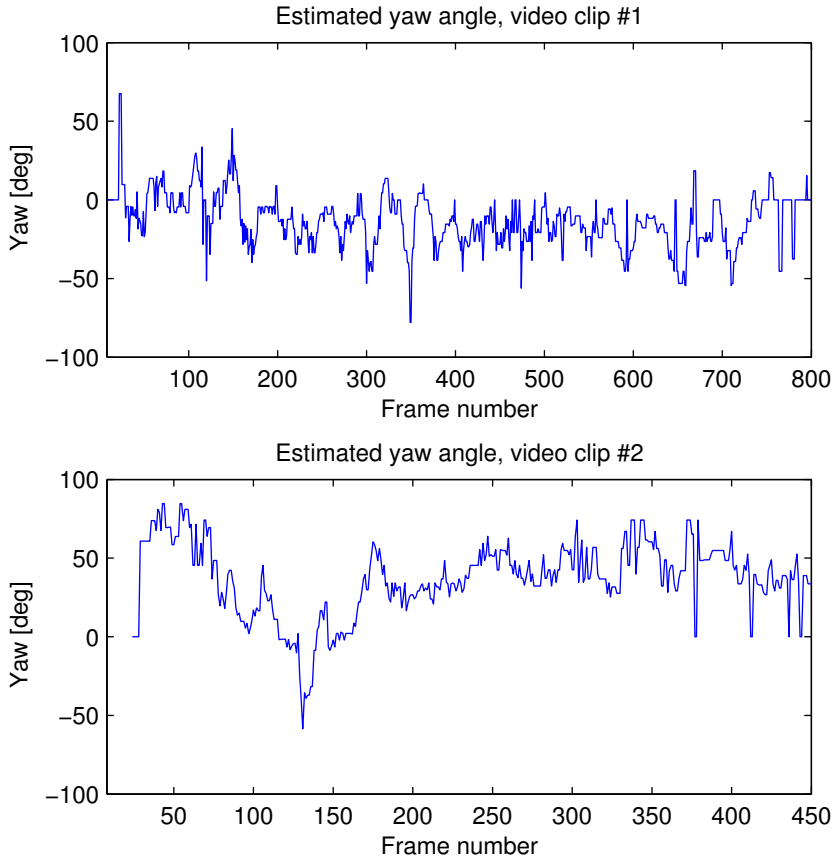


Figure 9.13: Yaw angle estimation for video clip # 1 and 2. The actual angles were roughly estimated to lie around -10 and 30 degrees for clip # 1 and 2 respectively.

9.2.4 Angle estimation

Yaw estimation was done using equation (7.14) from section 7.4. Distance estimations from the previous section were used as distance, d, in the equation, and the same focal length, f = 0.33, was used. The actual distance between the hub center and the top of the tower was assumed to be $|p_1^w - p_2^w| = 5m$.

Actual yaw angles are visually estimated to lie around -10 degrees for video clip # 1 and close to 30 degrees for video clip # 2. The results are shown in figure 9.13. One observes the mean value for each does not lie too far from the expected areas, but there is significant amounts of noise.

An essential cause for the high sensitivity is that the actual distance between the hub center and the top of the tower is low compared to the distance

from the hexacopter and the measured length, δ. Since the lowered threshold values for the Canny detector were used, the hub center estimation was less accurate, affecting the measurement of δ which relies on this value. By in addition having the noisy distance estimation as basis for the calculation, the inaccuracy was not surprising.

9.3 Computation Time

The computational demands of the program will here be examined. To get a notion of which procedures require the most computation power, the runtime of each major procedure was tracked. The procedures which were examined are the Canny edge detector, the Hough transform, downsampling and logic. In this context, logic refers to everything which is done after the Hough transform to recognize the wind turbine and estimate position, distance and yaw angle. The runtime for these procedures were obtained by measuring tick counts between selected checkpoints in the program.

9.3.1 Desktop analysis

For convenience some aspects of the program were first tested on a desktop computer. Specifications for the computer are given in table 9.3.

Figure 9.14 shows execution time when video clip # 1 was used as input. The execution time was mostly around 22-28 ms for each iteration, which is around 36-45 Hz. Due to the high resolution of the GoPro camera which produced the clip, the downsampling part (from 1920x1080 to 480x270) requires a significant amount of time. However, downsampling is heavily preferred over running the program in full resolution (as can be seen from figure A.1 in appendix). In either way, it shows that a high display resolution has a negative impact on the overall performance, and the best option is to use a camera which supports to readily provide video of lower resolution. Neglecting downsampling, the execution frequency lies around 53-77 Hz.

Among the remaining parts one observes that the Hough transform is the most demanding procedure. In addition, it is the most sensitive part. This

Table 9.3: Specifications for the desktop computer.

Processor	Intel Core i5-2500 @ 3.30GHz×4
Memory	7.9 GiB
Graphics	Gallium 0.4 on AMD CAICOS
OS	Ubuntu 13.10 32-bit

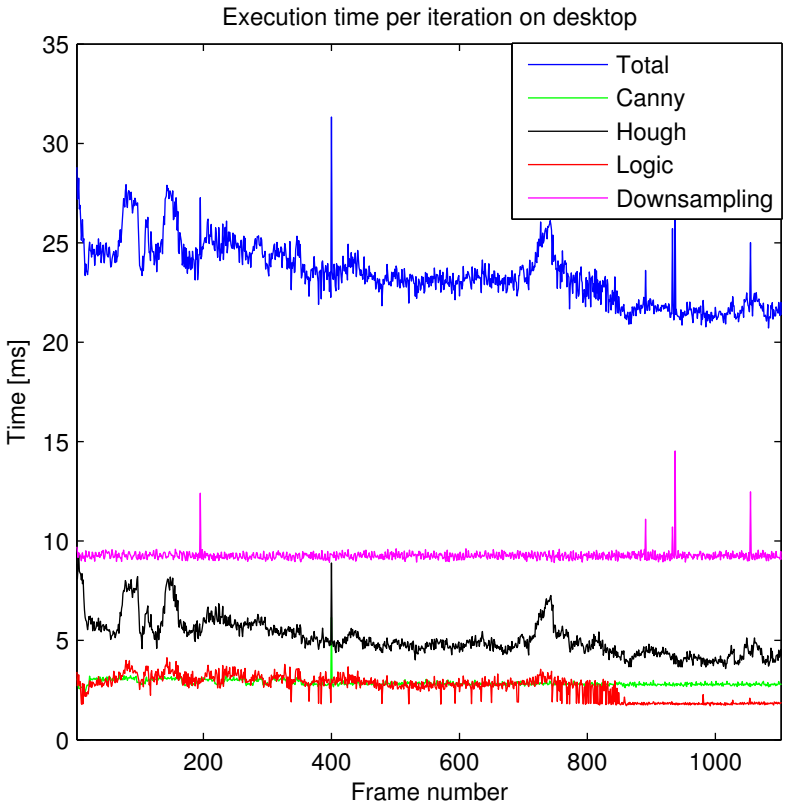


Figure 9.14: The execution time per iteration for the different parts of the program when run on the desktop computer, using the high resolution video clip # 1 as input. Therefore much time is used on downsampling. Otherwise the Hough transform is most significant.

can be seen by for instance the variations around the 100th frame. What happened there was that the propellers and frame of the hexacopter covered especially big areas of the image, which produced more Hough lines. This shows that the runtime of the Hough transform is significantly affected by the amount of limes. The logic part, however, is only mildly affected by this, while the increased amount of contours in the image seemed to be negligible impact on the Canny edge detector.

In figure 9.14 the Canny hysteresis threshold values were set as 400 and 200 for high and low threshold, respectively. In figure A.2, execution times are shown for a threshold values lowered to 300 and 200 for high and low threshold, respectively. This produces more edges in the edge map, and as expected causes an increase in computation time for the Hough transform. Surprisingly, the effect on the Canny runtime itself and the runtime for the logic part were mild by comparison. However, since the Hough transform (disregarding downsampling) has the highest cost, here even more so, it shows that one should be careful with lowering the threshold values too far.

9.3.2 Execution time on PandaBoard

Next the computation time was examined on the PandaBoard (specifications were given in section 5.1.3. Camera input was now used instead, to conform to the actual system setup. The target was still the wind turbine from video clip # 1, but now obtained by using the camera to record the video clip which played on the computer screen. Otherwise the same settings were used as in the first desktop test (figure 9.14).

The results are shown in figure 9.15, and a periodicity mostly in the range of 90-140 ms which translates to around 7-11 Hz. If the downsample time from the desktop computer is disregarded (subtracted) from the total time (resulting in 53-77 Hz), the results show that execution time for the Pandaboard is generally around 7-9 times higher. However, different image properties from the camera could have affected these numbers. Nevertheless, it is clear that the Hough transform still accounts for the majority of the computational effort. This means the computer vision program cannot be expected to run at a frequency much higher than 11 Hz on the Pandaboard. It also shows that with the current set-up the Canny threshold values should preferably be carefully based on lighting conditions, in order to prevent an excessive amount edges for the Hough transform.

Since the camera was configured to deliver frames at a low resolution, there was no need for downsampling. From the desktop results, it was observed that downsapling time (from 1920x1080) was even higher than the runtime of the Hough transform. If this was to be done on the Pandaboard,

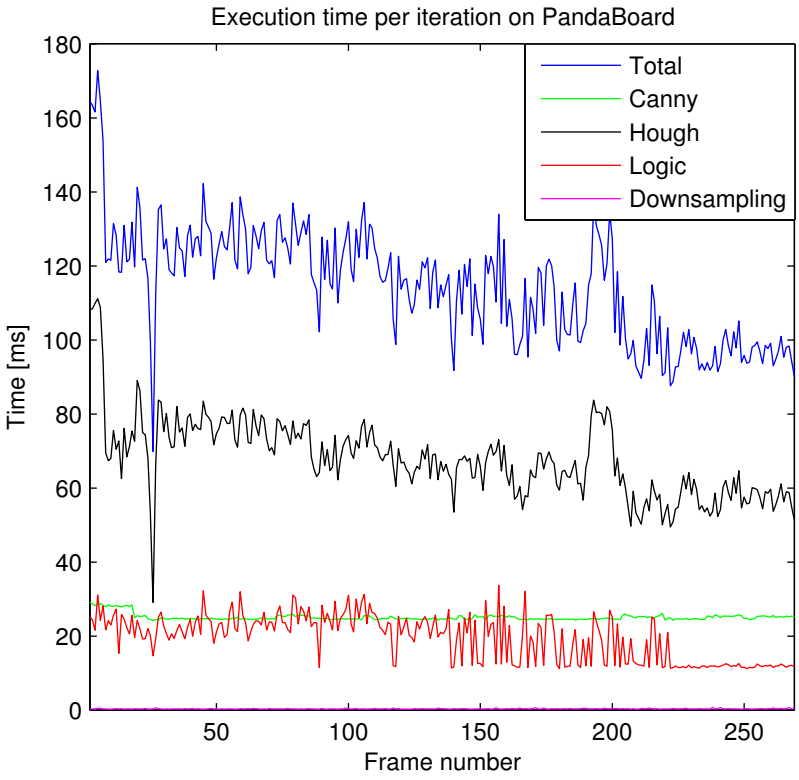


Figure 9.15: The execution time per iteration for the different parts of the program when run on the Pandaboard, using camera as input. The Hough transform has the highest computation demands.

the execution time could easily be doubled making the frequency fall down to the 5 Hz range, which shows the importance of having a low resolution input.

9.4 Flight control performance in relation to computation time

From the previous section it was found that the program cannot be expected to run faster than at about 10 Hz iteration frequency on the PandaBoard. In addition, this means that there is a 0.1 s delay from when something happens until it has been registered by the camera, analysed in the program and sent as an output reaction to the actuators. The purpose of this section is to examine whether this delay is sufficiently small to provide reliable control for

the hexacopter.

The test was run on an indoor UAV-lab (appendix B.1). For the test the hexacopter was set up to use computer vision to maintain a stable y^b-position (in the body-frame) in front of a plank placed vertically along the wall, using the camera described in section 5.1.4. Distance in the x^b-direction from the plank was about 2 meters. A stripped down version of the computer vision program was used, which simply tracked vertical lines in the image. This means that the of the logic part of the program was greatly reduced, but this had a small effect since this part initially required relatively small computation power, as was seen in figure 9.15. In addition, the PandaBoard was also running dune, but the effect on iteration frequency was small. Eventually, during flight the program proved to mostly maintain iteration periodicity between 0.10 s and 0.13 s.

For the test, only left/right movement i.e. roll was controller by the computer vision program. Due to the physical properties of the hexacopter, applying control in other dimensions can also transfer to left/right velocity. Thus, the roll was isolated in this way in order to make the relation between the camera stream and roll reaction as direct as possible. The movement in the other dimensions were limited manual control for pitch and yaw (at best effort) and by a sonar based altitude controller.

The roll controller was implemented as a simple PD-controller. Input was received from the computer vision program containing a position displacement length corresponding to the horizontal pixel distance, d_x, from the image center to the plank. This length was specifically given as $e_x = 0.01 \cdot d_x$. Additionally, right/left velocity was estimated from the IMU, given as v_y [m/s]. Using y as desired roll, and the gains K_p, K_d, the PD-controller was then given as follows:

$$y = Kp \cdot e_x + Kd \cdot v_y \tag{9.1}$$

The gains $K_p = 0.13$ and $K_d = 0.13$, were experimentally to work well and produced the behaviour shown in figure 9.16. It shows that the hexacopter oscillated slightly around the plank position, and the desired roll transfers well to measured roll. Except at the very start and end, the hexacopter maintained a roll angle between -0.1 and 0.1 radians and a velocity between -0.6 m/s and 0.6 m/s. The spikes at the beginning were due to an imbalance in the hexacopter which caused an uneven take-off, while manual landing is initiated at 62 s.

The oscillations in the system are caused by a number of factors. There was some turbulence due to the lab being indoors. In addition, there was some noise in the detection of vertical lines, as seen in the measured distance from the image center to the plank, due to some minor contours in the

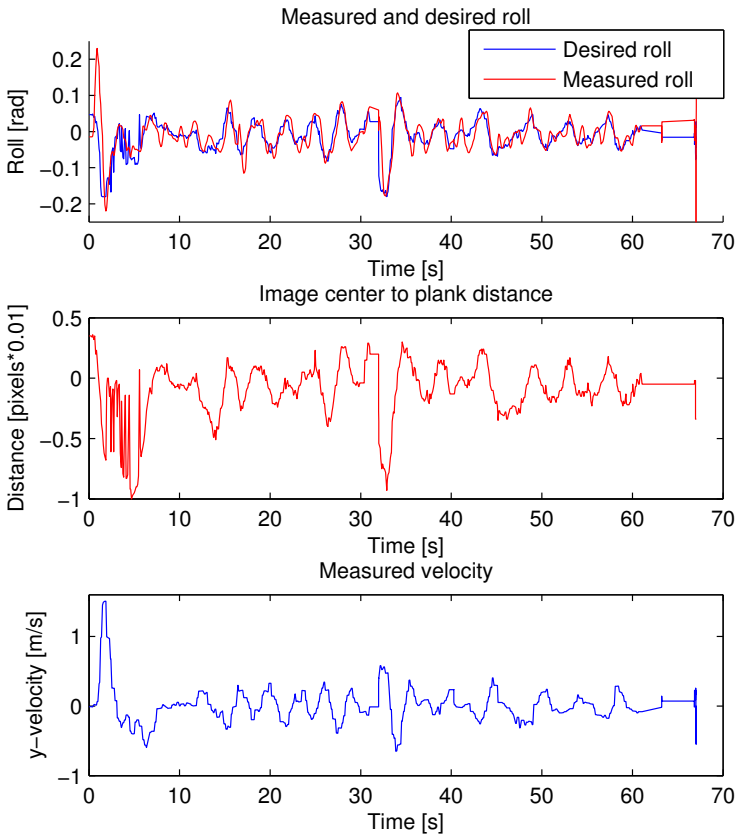


Figure 9.16: The resulting behaviour when attempting to follow the plank via roll control. The hexacopter oscillated slightly around the plank position.

background. Lastly, there is obviously the system delay which was the main purpose of this test. The delay and iteration frequency seem to lie inside the bounds for reliable control in this case, although the system is anticipated to struggle at further increased computational demand.

It should be noted, however, that since the hexacopter was hovering at about 2 meters from the plank, small movements caused significant shift in plank position in the image. At greater distances, as in a wind turbine inspection scenario, the target's position in the image will be less sensitive to UAV movement, giving more room for the controller.

A video of a roll control test flight with this setup is given in the file "roll.mp4" included in the digital appendix.

Chapter 10

Discussion

10.1 Sources of error

While the main sources of error were identified in section 9.1, a more in depth discussion will now follow.

The shadows in the video were responsible for most of the unfortunate effects in results. Tower lines did not reach the very top of the image, causing a lowered center for the blade search region, and the right tower edge was inaccurately detected for distance estimation. The reason why shadows caused these disruptions is essentially because the Canny edge detector was not sensitive enough to detect edges in these areas. Especially if the background also was dark in this area, the local intensity gradient in the image would be too low for an edge to be identified. A detection could be forced if the threshold values for the detector were lowered, but in the worst cases that would be to an excessive point resulting in an edge map filled with disturbance.

To omit this issue without utilizing a completely different approach a suggestion could be to utilize Hough circles. Instead of using the highest point among the tower lines as center for blade search, one could search for a circle further in the direction pointed by the tower lines which could identify the hub. If successful, this would in turn allow using a smaller search radius, and less false positives would be detected. Furthermore the radius of the detected Hough circle could then have been used to provide a measured width which could have been a more reliable base for distance estimation. The main reason Hough circles were not integrated in the algorithm is because of computational demand. Such an algorithm as proposed would require both Hough lines and Hough circles to be run in the same iteration. Not only does the Hough lines and Hough circles require high computational costs by themselves (as was shown for Hough lines in 9.3), but in addition,

as implemented in OpenCV they rely on different types of edge maps and would thus require double edge map calculation. This would presumably bring the iteration frequency past or right at the edge of bounds for reliable control.

The horizon proved to interfere with the accuracy by creating falsely detected blade lines. Due to the various restriction this was prevented to happen for most circumstances, but not when the horizon was too close to the hub. A quick solution could be to remove all horizontal lines, but this would obviously risk removal of any blade which would happen lie horizontally. Another more sophisticated and suggested approach would be to first identify the horizon by searching for horizontal lines spanning the entire image, before removing blade lines which were aligned at the same height. In this case, differentiation between the blade and the horizon would only be an issue in the specific case where a blade lies both horizontally and appears at the same level as the horizon in the image. However, since the lines then are aligned it would not be a problem anymore, because the intersection points with the other blades would be the identical and thus the estimated position would not be disordered. Unfortunately, this method was added as it was not easily applicable on the current test data, because of the distortion caused by the GoPro camera which makes the horizon curved.

As with most image processing methods the result changes along with the chosen parameters. The Canny edge detector is the bottleneck in this regard, since the rest of the program is mostly based on geometric properties rather than environment conditions. Nevertheless, the program proved to behave consistently over a relatively wide range of parameters, which suggests that the susceptibility to environment, weather and lighting changes is of lesser extent. If it were to be used under significantly different condition in practice, one could consider designing a mapping from lighting intensity to a parameter set or simply set up fitting weather profiles. A more sophisticated approach could utilize adaptive techniques, but based on the testing results this was not deemed necessary for further investigation.

10.2 Performance evaluation

Position estimation proved to work well at long distances, while losing accuracy at closer range. With the Kalman filter tracking enabled, the estimation was also fairly accurate at closer ranges, provided a successful detection was in fact made. The amount of successful detections followed a similar pattern. When reducing the distance, the range of y-positions providing successful detection was gradually limited. If the main issues can be resolved, as men-

tioned in the previous section, the performance can be expected to improve considerably at closer distances.

For distance and yaw angle estimation, the results were not too far to the expected area but there was significant noise in the output. The main problem was to find easily recognizable key points for measuring lengths in the image. Using the tower width proved difficult due to the shadow along the edge which caused a too narrow measurement and thus an exaggerated estimate. If a more reliable method is found to identify key points or a specific length in the image, it is expected provide more reliable estimates. Using Hough circles to identify the diameter of the hub could be a possible solution.

On the other hand, among the main reasons to utilize distance estimation in the first place, was to detect when the UAV arrived at the desired distance. At this range, if some key points were in fact recognized, the estimated position would be rather accurate since the size of objects are inversely proportional with distance. However, this distance would also be an attractive range to use a sonar, which would avoid the issue altogether.

Yaw angle estimation also suffers from the key point recognition issue. The approach which was presented utilized key points which were already provided by the steps required for position estimation. Since the points were fairly close to each other compared to the distance from the wind turbine, it was susceptible to noise, a was further affected by relying on noisy distance estimation.

If some points of known actual length were easily recognizable, such as the tips of the wind turbine, the yaw orientation could be estimated in the fashion described in 7.4.1. The tips are however outside of view, and the remaining parts of the wind turbine lack features.

Some more general information can however be obtained from the yaw angle estimation. The result form video clip # 2, for instance, (figure 9.13) showed that a tendency towards a high positive angle, which matched the fact that the actual angle was around 30 degrees. By using a low pass filter one would have observed a sustained high angle measurement, and could thus have made a meaningful decision reduce the angle. The purpose was, however, to decide the movement direction needed to minimize the angle, rather than calculating the exact angle. Therefore, by further improving the distance estimation, and by utilizing a deadband in conjunction with a low pass filter, it might be used as a basis for a decision.

10.3 Method Evaluation and Additional Suggestions

In overall the Hough line transform provided a robust way to extract the contours of the wind turbine, when not disrupted by shadows. Additionally, the implementation of the probabilistic Hough transform in OpenCV is efficient and the remaining parts of the algorithm has low computational demand. Therefore, the algorithm was able to run efficiently enough to provide a reasonable runtime iteration frequency on the PandaBoard.

The evident trend of diminishing performance at reduced distance reflects the fact that the program was primarily designed on the idea to capture the three-point star shape of the wind turbine. As the distance is reduced, this shape becomes less prominent in the image, and the various disturbances gain a greater influence.

On the other hand, at closer distance the circular shape of the hub gradually becomes dominant. For this reason, a suggestion could be to utilize the Hough circle transform. At long to medium distance, the Hough circle was determined too unpredictable. Up close, however, the hub is big enough to provide a stronger match. By also restricting the search to circles of considerable size the reliability improved drastically. Figure A.3 in appendix A shows a result where simple the center of the detected Hough circle is plotted. With some refinement this could be utilized in the program, for instance by switching from the current algorithm to a circle based detection method at an appropriate distance.

While the objective of the recognition algorithm was to provide necessary information for the UAV to find a suitable path for approach, the position data could also be used in a state observer scheme for better control. By the control layer of the system (see Høglund (2014)), the system was known to drift if movement in y-direction (sideways in the body frame) is merely based on integrated estimation from the accelerometer. Employing the position data for roll control could thus provide more stable control in y-direction, especially with presence of significant wind. Similarly, the estimated z-position could be used alongside the barometer to maintain stable height control. Other objects than a wind turbine can obviously be used as reference points, provided tracking can be performed reliably. An optical flow approach could be used in a similar manner, but the advantage of using a fixed object is that one can easier translate a movement to a specific length in reality, provided the distance to the object is estimated. A gimbal should also be used by utilization of such a method, because pitch rotation would change the z-position in the image and thus disrupting any mean-

ingful height data. Similarly a yaw rotation would disrupt any y-direction movement estimation.

10.4 Comments on system set-up

While camera based pitch and height control was not tested due to limited space at the UAV lab, there is concern regarding their performance because of how they affect each other. Since a pitch rotation tilts the camera along, both pitch and height determine the height at which the target appears in the image. However, in the video clips this was not an issue, because the camera was mounted on a gimbal which maintained a stable pitch orientation for the camera. Another benefit is that detection of vertical lines will not need to rely on roll data from the IMU to compensate for vehicle roll. For these reasons, it is suggested to consider adding a gimbal as part of the system set-up.

For a full scale operation, the flight time is expected to be a major issue. When the video clips were recorded, the hexacopter was able to keep itself airborne for a total of 5-6 minutes before running out of battery power. Moreover, the time required to reach the desired height and fly back down totalled at up to 3 minutes for each flight, leaving about 2 minutes of inspection time. It should be noted that it was windy, and under calmer conditions some minutes could be saved by less stabilization effort. Hopefully, by optimizing the hexacopter for the task and with better battery technology, flight time can be increased to a more feasible range for this mission.

10.5 Conclusion

In this thesis, a computer vision algorithm for recognition and tracking of a wind turbine has been presented. Based on knowledge from the tracking data a method was suggested for approaching the wind turbine for inspection. The wind turbine recognition algorithm was based primarily on the Hough line transform, due to its ability to capture the main features of the wind turbine and its low computational demand. The hub center position was estimated by analysing Hough lines and tracked using a Kalman filter. Distance and yaw angle were estimated by using the pinhole camera model and coordinate transformations.

Test video data was recorded at Bessakerfjellet wind farm. Running the program on the test data showed that the recognition and hub center tracking worked well at long distance. The performance deteriorated for shorter distances where the three-point star resemblance was less distinct. Distance and yaw angle estimation was inaccurate due to lack of strong recognizable lengths in the image. Computational demands were low enough to obtain a iteration frequency of 7-10 Hz on the PandaBoard. This proved to be adequate for stable control, although by a limited margin. If the addressed issues can be resolved, the program might be prepared for testing on wind turbine scale. The challenge remains, however, regarding the inspection of the individual blades, which was outside the scope of this work.

The findings might prove relevant for other missions involving tracking of objects where the Hough transform is preferred for recognition. Particularly for unmanned vehicles, the analysis on computational demand raises points which might be taken in consideration for decisions in both software design and hardware selection.

10.6 Future Work

The main sources of error described in section 10.1 should be addressed. Adding the Hough circle transform to the algorithm was suggested as a possible solution for better recognition of the hub. Investigating Hough circles in terms of performance and computational demand could therefore be a logical next step.

If computational requirements becomes an issue by addition of other methods, it could be considered to employ a scheme were the program performs a different task for each iteration. It was seen that by using Kalman filter aided tracking, the estimated position was smooth even when a significant amount of failed detections were present. Therefore, if another method

is performed in between, which for instance utilized another method specialized for distance estimation, the position can still be obtained from the predicted estimation.

Pitch, yaw and height control also remain to be tested for the system. Both pitch and height control were deemed to unreliable to test inside the confined space of the UAV lab without gimbal support. In addition, more accurate distance estimation should be obtained before attempting camera based pitch control.

Investigating distance estimation through other means could be considered. A laser rangefinder (LFR) could be used for the purpose. The LightWare SF02 (suggested by Høglund (2014)) might be a good choice, since it is light and compact and allegedly operates well in the relevant 0-40 meter range. Since the program's position estimation was accurate at long ranges, it could be possible to target the laser onto the wind turbine from these distances.

Appendix A

Supplementary Results

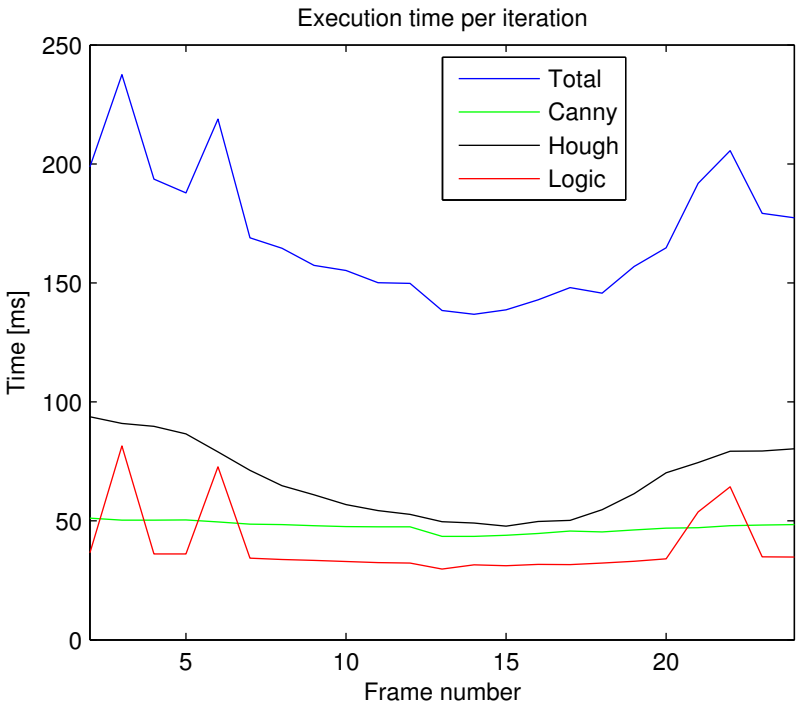


Figure A.1: Execution time per iteration on the desktop computer using a high resolution (1920x1080) video as input. Downsampling is not applied, and thus the computational demand is drastically increased.

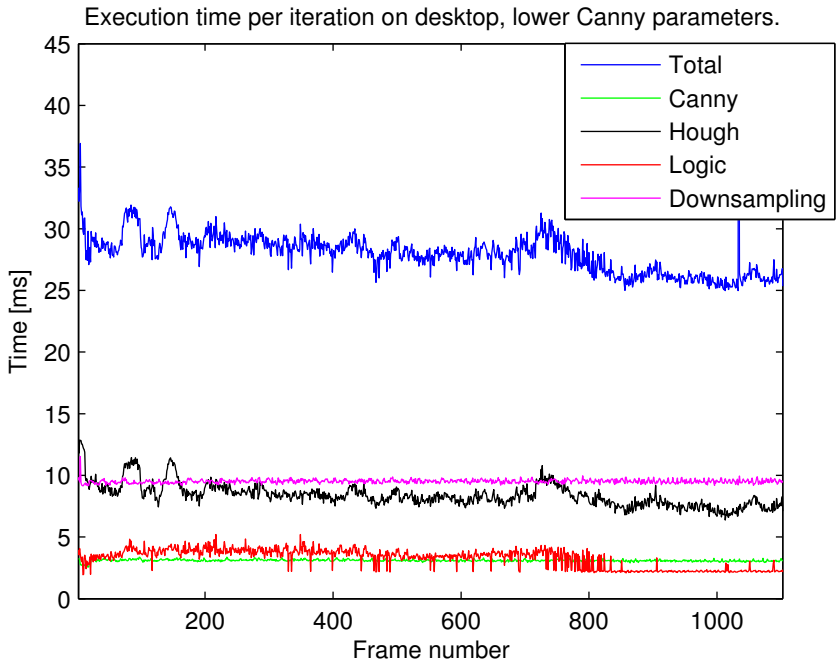


Figure A.2: Execution time per iteration on the desktop computer. The canny hysteresis threshold values were lowered, which due to a denser edge map caused an increase in processing time for the Hough transform.

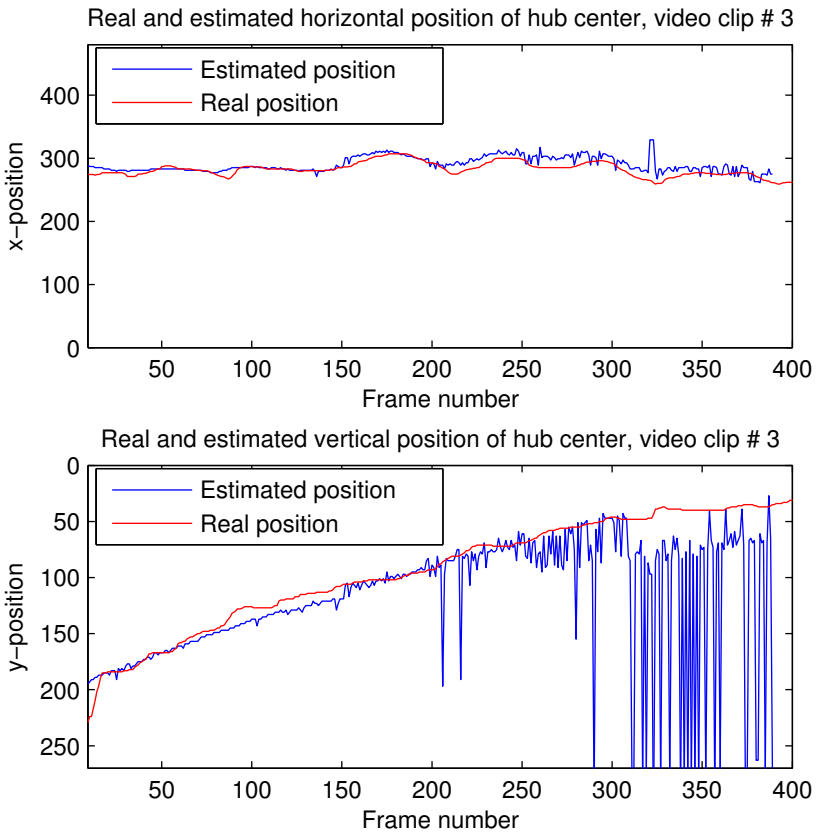


Figure A.3: Using Hough circle transform for position estimation of the hub center. The estimated point is simply obtained from the center of the detected circle. Imposed limits for circles were maximum radius = 70 pixels, minimum radius = 50 pixels. Minimum gap between detected circles was set to an amount much bigger than display resolution so that only a single circle is obtained.

Appendix B

Locations

B.1 UAV lab

The UAV-lab is located at the roof and built of wood to providing strong GPS signals. Inside a 5x5x3 meter net is placed to protect the hexacopter and spectators from damage in case something goes wrong.

Figure B.1: UAV-lab

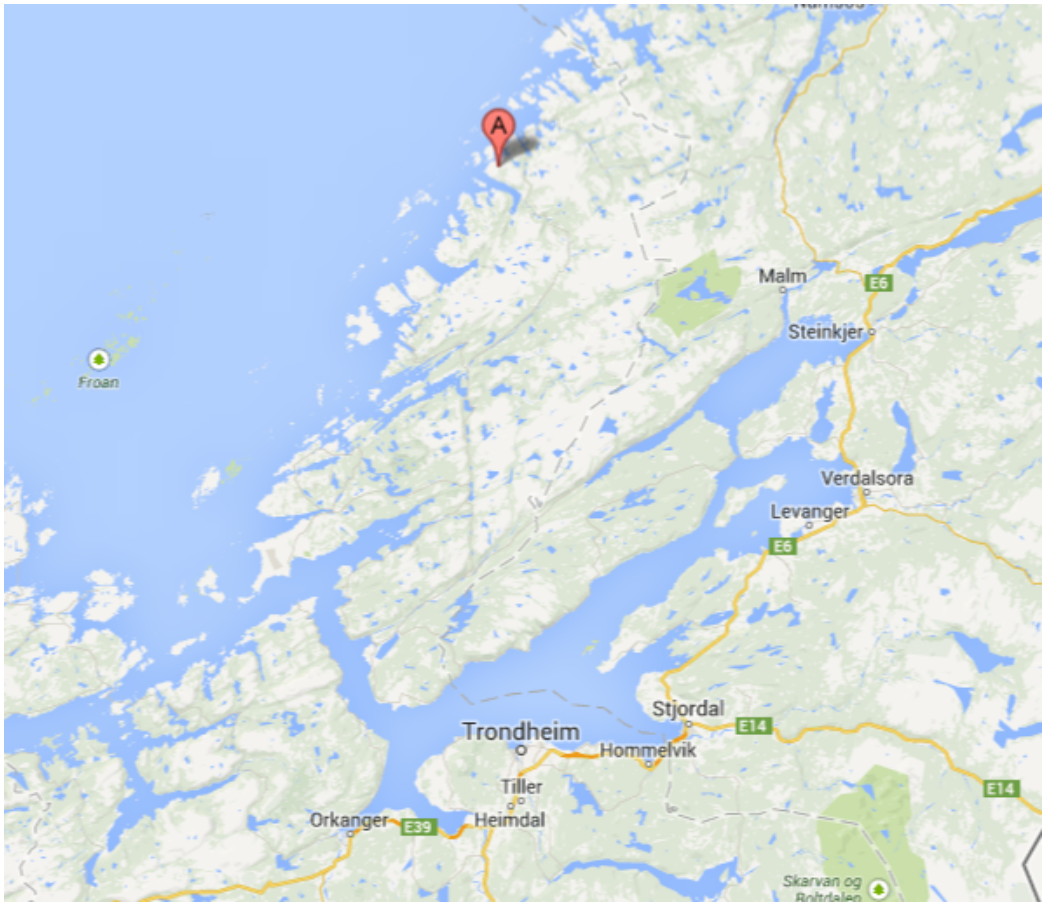


Figure B.2: The marker shows the location of Bessakerfjellet wind farm. Source: Google Maps.

B.2 Bessakerfjellet wind farm

The wind farm at Bessakerfjellet is owned and run by TrønderEnergi, and consists of 25 wind turbines. It is located about 100 km north of Trondheim. The wind turbines which are in use are of the type E-70, delivered by Enercon (ENERCON, 2014).

Appendix C

Hardware

C.1 GoPro HERO3+ Black Edition

When gathering video data at the wind farms, a GoPro HERO3+ Black Edition was used.

Figure C.1: GoPro HERO3+ Black Edition

Table C.1: Specifications for GoPro camera

Video mode used in project	
Video resolution	1920×1080
Frame rate	24
Field of view	69.5°×118.2°
Physical properties	
Weight	74g
Weight with housing	136g
Dimensions	42mm×60mm×30mm

Bibliography

Peter J. Birt. Fast filter transforms for image processing. *Computer Graphics and Image Processing*, 16(1):20–51, 1981.

F. Caballero, L. Merino, J. Ferruz, and A. Ollero. Vision-based odometry and slam for medium and high altitude flying uavs. In KimonP. Valavanis, Paul Oh, and LesA. Piegl, editors, *Unmanned Aircraft Systems*, pages 137–161. Springer Netherlands, 2009. ISBN 978-1-4020-9136-0. doi: 10.1007/978-1-4020-9137-7_9. URL http://dx.doi.org/10.1007/978-1-4020-9137-7_9.

John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, Nov 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851.

Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, January 1972. ISSN 0001-0782. doi: 10.1145/361237.361242. URL http://doi.acm.org/10.1145/361237.361242.

e-con Systems. e-cam51_usb, 2014. URL http://www.e-consystems.com/5mp-usb-cameraboard.asp. Accessed 13 Jun 2014.

ENERCON. E-70 wind turbine, 2014. URL http://www.enercon.de/en-en/61.htm. Accessed 14 Jun 2014.

Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

Sondre Høglund. Autonomous inspection of wind turbines and buildings using a uav. Master's thesis, NTNU, 2014.

P.V.C. and Hough. Machine Analysis Of Bubble Chamber Pictures. *Conf.Proc.*, C590914:554–558, 1959.

J. Illingworth and J. Kittler. A survey of the hough transform. *Computer Vision, Graphics, and Image Processing*, 44(1):87 – 116, 1988. ISSN 0734-189X. doi: http://dx.doi.org/10.1016/S0734-189X(88)80033-1. URL http://www.sciencedirect.com/science/article/pii/S0734189X88800331.

R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME - Journal of Basic Engineering*, (82 (Series D)):35–45, 1960. URL http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf.

Farid Kendoul, Isabelle Fantoni, and Kenzo Nonami. Optic flow-based vision system for autonomous 3d localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, 57(6-7):591–602, 2009. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/j.robot.2009.02.001. URL http://www.sciencedirect.com/science/article/pii/S0921889009000396.

Carolyn Kimme, Dana Ballard, and Jack Sklansky. Finding circles by an array of accumulators. *Commun. ACM*, 18(2):120–122, February 1975. ISSN 0001-0782. doi: 10.1145/360666.360677. URL http://doi.acm.org/10.1145/360666.360677.

N. Kiryati, Y. Eldar, and A.M. Bruckstein. A probabilistic hough transform. *Pattern Recognition*, 24(4):303 – 316, 1991. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/0031-3203(91)90073-E. URL http://www.sciencedirect.com/science/article/pii/003132039190073E.

Frederik S. Leira. Infrared object detection & tracking in uavs. Master's thesis, NTNU, 2013.

R.C. Leishman, T.W. McLain, and R.W. Beard. Relative navigation approach for vision-based aerial gps-denied navigation. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 343–352, 2013. doi: 10.1109/ICUAS.2013.6564707.

Yu-Chi Liu and Qiong-Hai Dai. A survey of computer vision applied in aerial robotic vehicles. In *Optics Photonics and Energy Engineering (OPEE), 2010 International Conference on*, volume 1, pages 277–280, 2010. doi: 10.1109/OPEE.2010.5508131.

D. Magree, J.G. Mooney, and E.N. Johnson. Monocular visual mapping for obstacle avoidance on uavs. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 471–479, 2013. doi: 10.1109/ICUAS.2013.6564722.

J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, 78(1):119 – 137, 2000. ISSN 1077-3142. doi: http://dx.doi.org/10.1006/cviu.1999.0831. URL http://www.sciencedirect.com/science/article/pii/S1077314299908317.

L. Mejias and D. FitzGerald. A multi-layered approach for site detection in uas emergency landing scenarios using geometry-based image segmentation. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 366–372, 2013. doi: 10.1109/ICUAS.2013.6564710.

Goddard Space Flight Center NASA. Globe east, February 2002. URL <http://eoimages.gsfc.nasa.gov/images/imagerecords/57000/57723/globe_east_540.jpg>. Accessed 26 Nov 2013.

PandaBoard. Pandaboard es platform, 2014. URL http://pandaboard.org/content/platform. Accessed 10 Jun 2014.

3D Robotics. Arducopter 3dr hexa b, 2014. URL http://store.3drobotics.com/products/arducopter-3dr-hexa-b-1. Accessed 10 Jun 2014.

K. Schauwecker and A. Zell. On-board dual-stereo-vision for autonomous quadrotor navigation. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 333–342, 2013. doi: 10.1109/ICUAS.2013.6564706.

SharkD. Hsv color solid cylinder alpha lowgamma, March 2010. URL <http://en.wikipedia.org/wiki/File:HSV_color_solid_cylinder_alpha_lowgamma.png>. Accessed 07 Des 2013.

Alvy Ray Smith. Color gamut transform pairs. *SIGGRAPH Comput. Graph.*, 12(3):12–19, August 1978. ISSN 0097-8930. doi: 10.1145/965139.807361. URL http://doi.acm.org/10.1145/965139.807361.

Stephen M. Smith and J. Michael Brady. Susan—a new approach to low level image processing. *Int. J. Comput. Vision*, 23(1):45–78, May 1997. ISSN 0920-5691. doi: 10.1023/A:1007963824710. URL http://dx.doi.org/10.1023/A:1007963824710.

I. Sobel and G. Feldman. A 3x3 Isotropic Gradient Operator for Image Processing. Never published but presented at a talk at the Stanford Artificial Project, 1968.

S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985. ISSN 0734-189X. doi: http://dx.doi.org/10.1016/0734-189X(85)90016-7. URL `http://www.sciencedirect.com/science/article/pii/0734189X85900167`.

M. Williams, D.I. Jones, and G.K. Earp. Obstacle avoidance during aerial inspection of power lines. *Aircraft Engineering and Aerospace Technology*, 73(5):472–479, 2001.

Li-hui Zou, Jie Chen, Juan Zhang, and Li hua Dou. The comparison of two typical corner detection algorithms. In *Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium on*, volume 2, pages 211–215, Dec 2008. doi: 10.1109/IITA.2008.275.