NTNU – Trondheim
Norwegian University of
Science and Technology

Infrared Object Detection & Tracking in UAVs

Frederik Stendahl Leira

Master of Science in Engineering Cybernetics
Submission date: June 2013
Supervisor: Thor Inge Fossen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

The present thesis describes the design and implementation of a small, light weight and power efficient payload system for the use in *unmanned aerial vehicles* (UAVs). The primary application of the payload system is that of performing real-time object detection and tracking based on an *infrared camera*. The implemented object detection algorithm utilizes pre-trained *classifiers* to perform detection, and the implemented object tracking algorithm is based on an *estimate-and-measure* tracking approach. The estimator used is a standard *Kalman filter*, which assumes a *linear motion model* for the tracked objects. A *global nearest neighbor* approach was used to match the measurements to the tracked objects. Two types of classifiers were trained. The first classifier was trained with a *support vector machine* in combination with the *histogram of oriented gradients* feature representation. The second classifier was created by constructing a *boosted cascade* of classifiers trained using the *AdaBoost* algorithm by means of a set of *Haar-like features*. Furthermore, experiments are presented which demonstrate that the system is able to consistently track humans in many simulated real-time scenarios. However, it was found that in the presence of abrupt and relatively large object displacements, the linear motion model was not sufficiently accurate to keep track of the object. This implies that the tracking algorithm can benefit from the implementation of a non-linear estimator. Finally, the payload was found to be able to perform simulated real-time tracking, while at the same time performing several additional tasks. This includes sending important data to a control station located on the ground, and also simulating control over the UAV.

Keywords: unmanned aerial vehicle, classifier, infrared camera, object detection, object tracking, Kalman filter, linear motion model, global nearest neighbor, histogram of oriented gradients, support vector machine, boosted cascade, Haar-like features.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree MSc. at the Norwegian University of Science and Technology.

I would like to thank Professor Thor Inge Fossen and Professor Tor Arne Johansen, my supervisors, who trusted me to take on this project. They have given me very valuable and constructive suggestions during the development of this research work. Thanks also to Dr. Esten Groetli for guiding me in the right direction regarding software solutions.

Moreover, I would like to thank all the people at Maritime Robotics for helping me in the process of finding appropriate hardware suitable for my project, as well as giving me the opportunity to test their infrared camera system.

Finally, I would like to thank my parents, and Synne, for their support.

Trondheim, June 24, 2013

Frederik Stendahl Leira

Problem Text

The main objectives of the MSc project are to develop a sensor payload for infrared (IR) object detection and tracking in unmanned aerial vehicles (UAVs). This includes the design, integration and implementation of electrical and software components of the payload.

The main components in the design of the payload should be the following

- **Processing unit (CPU)**
  The module should be able to handle the following functions
  - Control and configuration of payload sensors.
  - Interface to navigation module.
  - Data logging to local storage device.
  - Streaming of navigation and payload sensor data over IP radio / communication link.
  - IR-based Object detection and tracking.
- **Infrared camera module**
  The module should be suitable for real time processing with a suitable number of frames per second. It should also be able to perform data streaming to a control station located on the ground.
- **Video camera module**
  The module should be suitable for real time processing with a suitable number of frames per second. It should also be able to perform data streaming to a control station located on the ground.

- **Navigation module**
  The module should be able to output the UAVs attitude and position data to the processing unit and/or the local network in the payload.
- **Electric power supply**
  The module should be able to stably supply the payload with sufficient electrical power.
- **Communication module**
  The module should enable communication between payload modules. In addition, it should be able to communicate with a control station located on the ground.

The design should as far as possible be modular such that modules can be replaced without too much redesign. Ideally, there should be space and weight available to add one or two additional modules.

The ground control station will communicate over a wireless data link, and should be able to display UAV status, payload sensor images and processed information such as tracked objects.

The main tasks are

- Part 1: Payload
  - Design of the payload: Specification and choice of components, connection diagrams, data protocols, software and computer architecture. The design must satisfy requirements for size, weight and power usage. Use existing modules for CPU, communication, navigation, camera and battery as far as possible.
  - Design command station. Evaluate existing software and hardware together with Maritime Robotics.
  - Implement, build and laboratory test payload and command station.
  - Field test of the payload.
- Part 2: Object detection and tracking
  - Implement and evaluate algorithms for object detection based on IR sensor. Test the algorithms in the payload CPU.

  - Implement and evaluate an algorithm for object tracking based on IR detection. Test the algorithm in the payload CPU.
  - Implement data logging of the tracking results, video streaming and a simple vision-based navigation algorithm.
  - Evaluate the performance of the tracking algorithm in simulated real-time object detection and tracking tests.

- Part 3: Conclusions
  - Make suggestions for improvements to payload and architecture, as well as algorithms and systems for IR-based object detection and tracking applications.

Limitations:

1. The navigation algorithm is not required to control an UAV, but should be able to output navigation commands to an autopilot or a navigation control software.

This page intentionally left blank.

Contents

List of Figures

List of Tables

This page intentionally left blank.

List of Acronyms

BC Boosted Cascade

CPU Central Processing Unit

EO Electro-Optical

FPR False Positive Rate

FPS Frames Per Second

GPS Global Positioning System

GPU Graphical Processing Unitd

GNN Global Nearest Neighbor

HL Haar-like

HOG Histogram of Oriented Gradients

HTTP Hypertext Transfer Protocol

IMC Intermodule Communication

INS Inertial Navigation System

IR Infra-red

LoS Line of Sight

MJPEG Motion Joint Photographic Experts Group

NNSF Nearest Neighbor Standard Filter

OS Operating System

POE Power Over Ethernet

RAM Random Access Memory

ROC Receiver Operating Characteristic

ROI Region of Interest

SBC Single Board Computer

SoC System on Chip

SVM Support Vector Machine

TCP/IP Transmission Control Protocol/Internet Protocol

TPR True Positive Rate

UAV Unmanned Aerial Vehicle

URL Uniform Resource Locator

USB Universal Serial Bus

Chapter 1

Introduction

The topic of the present thesis is real-time object detection and tracking in unmanned aerial vehicles (UAVs), with the use of an infrared imaging camera. This chapter will give a brief background to this research field, and state some of the latest challenges and unsolved problems related to this topic. Furthermore, a problem formulation with associated goals are given, before the chapter ends with an outline of the thesis and how the problems will be solved.

1.1 Background

Unmanned aerial vehicles (UAVs) have been an active topic for research for several years [40]. They can be applied in a large variety of different scenarios, and supply a testbed to investigate several unsolved problems such as path planning, control and navigation. Furthermore, with the availability of low cost, robust and small video cameras, UAV video has been one of the fastest growing data sources in the last couple of years [44]. In other words, object detection and tracking as well as visual navigation has recently received a lot of attention. Some of the most recent work within this field include autonomous see-and-avoid systems and autonomous visual based landing [12, 17].

Several object detection and tracking algorithms suitable for the use in UAVs already exists, but most of them are restricted to the case of tracking *moving objects* [6, 28, 29, 35, 44]. Now, with infrared cameras becoming cheaper and more available to the public, the focus of target tracking in UAVs has shifted more towards the case of *search and rescue* operations. In such operations it can *not* be assumed that the tracked object (a human), is moving. On the contrary, during such operations it might be expected that most targets will be stationary more often than not. This introduces the need for novel detection and tracking algorithms which are not limited to tracking only moving targets.

A common approach to object detection and tracking in UAVs is to send the recorded video data to a ground station for processing [32]. Object detection and tracking is then performed at a high-end desktop computer, before command signals are sent to the autopilot located on-board the UAV. There are several complications associated with this approach. For example, for this to function properly, a reliable and fast wireless data connection is required between the UAV and the ground station at all times. If the UAV moves too far away from the ground station, the video signal is usually either transmitted with a huge lag time or worse, the data received at the ground station may be corrupted. This effectively limits the operational range of this approach drastically, and it is therefore in many cases not ideal.

Now, in recent years, computer hardware has become smaller, lighter, more power efficient *and* more powerful. This has lead to the possibility of implementing *real-time object tracking directly* on-board the UAV [8, 19, 33]. Performing the image processing on-board has several key advantages compared to sending the video to a control station, especially in search and rescue operations. The key advantages to this approach is that there is a reduction in the delay between the tracking and the resulting command signal sent to the autopilot. In addition, and arguably more importantly, this approach enables the UAV to travel outside the line of sight (LoS) of the control operator, effectively extending the range of operation.

The main focus of the present thesis is the design and implementation of a system that is able to perform real-time object detection and tracking *on-board the UAV*. A requirement for the tracking algorithm is that it should be able to track stationary as well as moving targets. Furthermore, the main video source for the tracking algorithm should be video captured from an infrared camera. Finally, it should be noted that there has been a recent increase in the resolution of infrared cameras, which effectively make them applicable for object detection over larger distances than previously possible.

1.2 Problem Formulation

The main objectives of the present thesis are to develop a sensor payload able to perform real time infrared object detection and tracking in UAVs. This includes the design and implementation of the electrical and software components of the payload.

The work of designing the total payload system is divided into the following three tasks

- **The payload**
  The design of the payload should include the specifications and choice of components, connection diagrams, data protocols and software choices. The design should as far as possible be modular, as well as satisfy requirements for size, weight and power usage.
- **Object detection algorithms**
  Promising methods for performing robust real-time object detection from UAVs with an infrared camera should be implemented and evaluated.
- **Object tracking algorithm**
  A tracking algorithm suitable for real-time tracking with an infrared camera should be implemented and tested.

For the implementation of the payload to be considered successful, it should be tested and verified to be able to

- Robustly detect and track humans in real-time with the use of an infrared camera. The tracking algorithm should be able to consistently track targets under a variety of different scenarios.
- Store the tracking results to the local storage unit.
- Stream processed images to a control station located on the ground.
- Supply the ground station with important information such as UAV status and battery life.

Furthermore, suggestions for improvements of the payload as well as the object detection and tracking algorithms should be made based on the test results.

1.3 Thesis Outline

To achieve the goals mentioned in the previous section, first a brief look on different approaches to the problem of object detection and recognition will be covered in chapter 2. Chapter 3 then proceeds to suggest an object tracking algorithm which can be used in conjunction with any object detection technique to achieve computationally efficient tracking. Chapter 4 subsequently cover the design of the total payload system.

Having designed the payload and reviewed suitable object detection and tracking algorithms for the use in UAVs, chapter 5 describes how all of this can be implemented and tested by describing a variety of different experiments. Chapter 6 then features a presentation and discussion of the most important results that were found during the experiments, before chapter 7 concludes the thesis with a brief summary of the work that was performed and the associated implications. Finally, in chapter 8, various ways to further develop and improve the payload system and its object detection and tracking algorithms are proposed.

Chapter 2

Object Detection and Recognition

Object detection in computer vision consists of finding a given object in an image. To be more specific, in our case we are concerned with the problem of telling whether or not a specific type of object (e.g a human) is present in an image or not. This is referred to as a *2-class classification problem* [10]. Now, the task of object detection is considered to be very challenging for computer systems in general. This is mostly due to the fact that the appearance of an object is dependent on many different factors. Just to mention a few aspects; viewing angle, scaling, illumination, occlusion, deformation and background clutter are all characteristics that play an important part in relation to how an object is projected onto the lens of a video camera. Hence, in the present thesis we want to use algorithms which are robust with respect to many of these factors.

The success of object detection systems is determined by two key factors: the feature representation and the learning algorithm. In other words, a good object detection system needs to have both a good feature representation *and* a good learning algorithm in order to have good performance. The feature representation is responsible for how to represent an object in an image in a reasonable way, using a reasonable amount of properties. The learning algorithm is responsible for the other half, i.e deciding whether the object is what we are searching for or not

(recognition). Such an algorithm is also referred to as a classifier. The following chapter provides a review of feature representations and classifiers.

2.1 Feature Representations

A feature in computer vision is used to denote a piece of information that is relevant to solving a given computer vision problem. In other words, features can be of almost any shape and color. Now, how and which features we choose to describe in an image is referred to as feature representation, and each instance of a feature representation is referred to as a *feature descriptor* [26]. This is illustrated with the following example. A circle in an image can be represented by two coordinates x, y and a radius r. This would then be a feature representation for the feature "circle". Now, say that a circle with radius r_0 is located at location x_0, y_0. The set r_0, x_0, y_0 would then be the *feature descriptor* of this circle. Furthermore, the *feature space* refers to the total span of the feature representation. In our example this would mean the combination of all possible circle locations and different radii. As previously mentioned, the choice of feature representation is crucial for the performance of object detection algorithms. Following is the review of two popular feature representations.

2.1.1 Haar-like Features

The principle behind Haar-like features is that instead of using the intensity value of pixels (as many feature representations do), the features make use of the change in contrast values between adjacent rectangular groups of pixels [31] [16]. The contrast variances between the rectangular pixel groups can in turn be used to determine relative light and dark regions. As seen in figure 2.2, Haar-like features are formed by combining two to three adjacent rectangular groups with a relative contrast variance. An important thing to note is that the rectangular regions can be independently scaled, effectively making the number of possible Haar-like features very large. Furthermore, the position of a Haar-like feature is defined relative to a fixed frame. This concept is illustrated in figure 2.1

Each Haar-like feature is represented by a scalar value, which is the difference between the sum of the pixel gray level values within the black rectangular regions

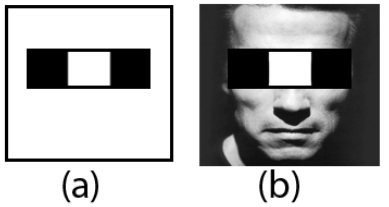

Figure 2.1: (a) shows a Haar-like feature and how it is defined relative to a fixed frame. (b) shows an image (a fixed frame) that has the characteristics of the Haar-like feature in (a).

and the white rectangular regions, i.e

$$f(\mathbf{x}) = \sum_{black\,rectangles} (pixel\ gray\ level) - \sum_{white\,rectangles} (pixel\ gray\ level) \quad (2.1)$$

where $\mathbf{x}$ contains all the pixels the Haar-like feature consists of. Now, using the operator f it is possible to search a region in an image, and estimate whether a Haar-like feature is present or not. Let us consider the case where we are given a Haar-like feature j with operator f_j. To check if a region $\mathbf{x}$ in the image has similar features to the Haar-like feature j, one can simply apply the function 2.1 to the region. This results in a *weak classifier*[1] $h_j(x)$, which can be defined as

$$h_j(x) = \begin{cases} 1 & if \quad p_j f_j(\mathbf{x}) \leq p_j \theta_j \\ 0 & otherwise \end{cases} \quad (2.2)$$

Where θ_j is a treshold value and p_j is a polarity indicating the direction of the inequality sign (e.g −1 or 1). The classifier h_j is equal to 1 if a region of interest ($\mathbf{x}$) is similar to Haar-feature j, and 0 otherwise. It should be emphasized that it is the operator f_j that decides which areas of $\mathbf{x}$ that should be treated as a black rectangle, and which areas of $\mathbf{x}$ that should be treated as a white rectangle. In other words, this is irrespective of the actual gray level value of a pixel in the image $i(x, y)$. How to use weak classifiers in order to make a strong classifier, i.e a classifier with strong correlation to the true classification, and how to set reasonable tresholds θ_j will be reviewed in section 2.2.2.

[1]The term *weak classifier* refers to the fact that this is a classifier that only works slighly better than random guessing [34]

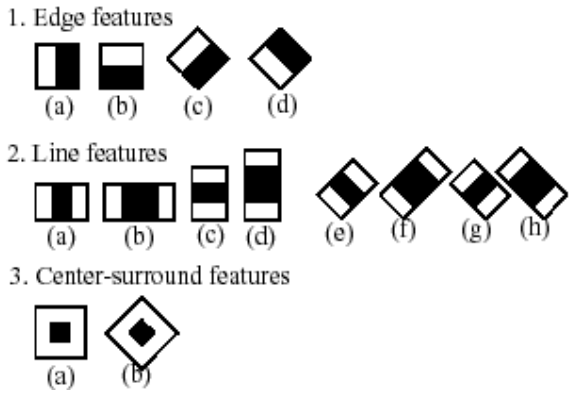

Figure 2.2: A list of different haar-features (image source: [27])

Now, a classifier utilizing Haar-like features may have to search a region for many different Haar-like features before it can conclude anything. This implies that the classifier will have to calculate the sum of pixel gray level values on possibly *thousands* of rectangles in *every* image for which object detection is wanted. Hence, a good way to calculate these sums is important for the speed and efficiency of Haar-like feature classification. This is achieved by calculating the *integral image*. The integral image of an image i is defined as follows [31]

$$I(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.3}$$

In short, letting $0, 0$ represent the top left corner of an image, this means that the integral image at location x, y contains the sum of the pixel gray level values above and left of x, y (inclusive) in the original image. This is illustrated in figure 2.3. Now, the reason for doing this is that it allows a computer to sum the pixel gray level values over a rectangle very efficiently. This can be seen from the following example. A rectangle consists of four corners described by four sets of x, y coordinates. If we let x_{TL}, y_{TL} represent the top left corner, x_{TR}, y_{TR} the top right corner, x_{BL}, y_{BL} the bottom left corner and finally x_{BR}, y_{BR} the bottom right corner, the sum of the pixel gray level values in this specific rectangle can be expressed in terms of the integral image as

$$\begin{aligned} \sum_{rectangle} (pixel\ gray\ level) = I(x_{BR}, y_{BR}) - I(x_{TR}, y_{TR}) \\ - I(x_{BL}, y_{BL}) + I(x_{TL}, y_{TL}) \end{aligned} \tag{2.4}$$

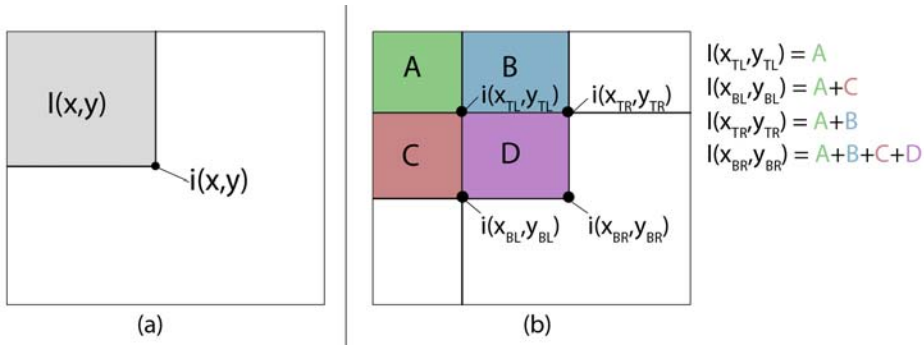

Figure 2.3: (a) shows how an image i is related to its integral image I. $I(x, y)$ is equal to the sum of the gray level values for all pixels in the gray rectangle. (b) shows the correlation between the corners of a rectangle in the original image i and the same points in the integral image I.

This is easily illustrated. Letting the above mentioned x, y corner coordinates represent the corners of rectangle D in figure 2.3, it is readily seen that $I(x_{TL}, y_{TL}) = A$, $I(x_{TR}, y_{TR}) = A + B$, $I(x_{BL}, y_{BL}) = A + C$ and $I(x_{BR}, y_{BR}) = A + B + C + D$. It should then be clear that inserting these equations into 2.4 yields the following

$$\sum_{rectangle} (pixel\ gray\ level) = A + B + C + D - (A + B) - (A + C) + A = D$$

This simple way of calculating the sum of a rectangle's pixel gray level values is one of the key advantages of Haar-like features over other feature representations. Exactly how Haar-features are used to create a good classifier is covered in section 2.2.2

2.1.2 Histogram of Oriented Gradients

The histogram of oriented gradients (hereafter denoted *HOG*) is a feature representation that can be used for robust object detection. This technique is based on the principle of counting occurrences of gradient orientations in subregions of an image [14]. That is to say, the HOG representation exploits the fact that objects within an image can be described by the vector field of gradients that they induce on the image. Mathematically, this is done by subdividing the image up into small connected regions (called *cells*), and then for each such cell computing a histogram

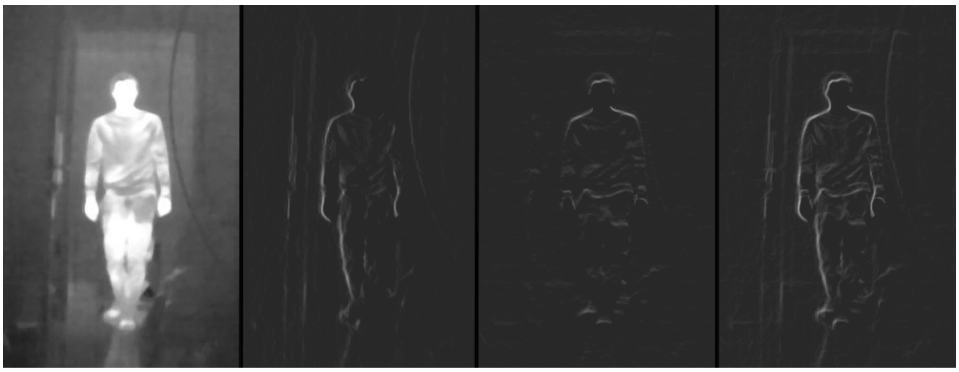

Figure 2.4: (a) shows a gray scale image captured from a thermal imaging camera and (b) shows the gradients in the horizontal direction. (c) is showing the gradients in the vertical direction and (d) is a merged version showing gradients in both the horizontal *and* the vertical direction.

of gradient directions (edge orientations) for the pixels within the cell. The combination of all histograms over all the cells represents the HOG descriptor.

To calculate a HOG descriptor in an image, the first operation we have to perform is to compute the magnitude and direction of the gradients of the image we want to analyze [14]. This can be achieved by calculating the gradients decomposed in the horizontal and the vertical direction. After computing this, finding the magnitude and direction of the gradients is just a matter of simple vector calculus. Now, in order to find gradients in the horizontal and vertical direction in an image, I, we can convolve/filter (see appendix A for details) the image with the simple 1-D masks $[-1, 0, 1]$ and $[-1, 0, 1]^T$, i.e

$$I_{grad_x} = I \star [-1, 0, 1] \qquad I_{grad_y} = I \star [-1, 0, 1]^T \tag{2.5}$$

The result of doing this can be seen in figure 2.4(b) and (c). Once the decomposed gradients are found, the magnitude ($G(x, y)$) and direction ($\theta(x, y)$) of a gradient at pixel x, y is easily calculated by the application of the following formulas [14]

$$\begin{aligned} G(x, y) &= \sqrt{I^2_{grad_x}(x, y) + I^2_{grad_y}(x, y)} \\ \theta(x, y) &= arctan(\frac{I_{grad_y}(x, y)}{I_{grad_x}(x, y)}) \end{aligned} \tag{2.6}$$

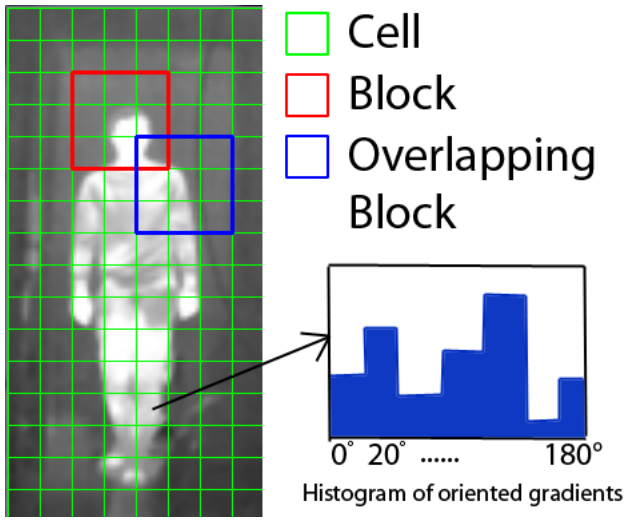

Figure 2.5: Each cell consists of $8x8$ pixels, and each block consists of $3x3$ cells. A histogram of oriented gradients is calculated for each cell in each block, based on the gradient magnitude and orientation for every pixel in the cell.

Once the gradient image is found, the next step is the orientation binning. This consists of iterating over all the pixels in a cell. Each pixel in the cell casts a weighted vote (based on the magnitude of the gradient at this pixel) for an orientation-based histogram channel (based on the gradients orientation at the particular pixel). The orientation-based histogram is divided into 9 bins evenly spaced over $0° - 180°$ ("unsigned" gradient direction). A small example of how this works is illustrated in figure 2.5

After finding the local histograms for each cell, there is one final step still remaining, which is to account for changes in illumination and contrast. This is done by locally normalizing the gradient strengths by grouping cells together into larger connected blocks. The cell histograms are then normalized with respect to their corresponding block. An important thing to note, is that for the best performance, tests indicate that these blocks should overlap [14]. This means that each cell contributes more than once to the final descriptor, with the only difference being which block it is normalized with respect to. The total HOG descriptor is represented as the vector of normalized cell histograms from every block. An illustration of a HOG descriptor for an image can be seen in figure 2.6. As a side note, the cells

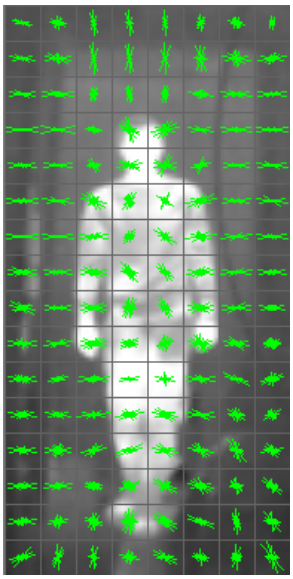

Figure 2.6: The figure shows a visualization of the HOG descriptor for an image. Each cell has a histogram of oriented gradients, visualized by showing all possible gradient orientations and scaling them appropriately, i.e the dominant gradient orientations has a larger magnitude than the less dominant orientations.

and the blocks can be represented by different shapes. Both *r*ectangular (R-HOG) and *c*ircular (C-HOG) [14] shapes can be used, and which works best depends on the application. In this thesis the implemented HOG algorithm uses rectangular cells, i.e is a R-HOG implementation.

Calculating a histogram of oriented gradients for each cell in every block, in every region that is to be searched can be very time consuming. To simplify this calculation, a similar trick to that of integral images in section 2.1.1 can be applied. The *integral histogram representation* is based on the same simple idea. Let us consider a histogram $h(x, y, b)$ where b is the channel/orientation, i.e $h(x, y, b)$ is the weighted contribution of pixel x, y to histogram channel/orientation b. Then the integral histogram representation for channel b can be described as

$$H(x, y, b) = \sum_{x' \leq x, y' \leq y} h(x', y', b) \tag{2.7}$$

It is readily seen that the same procedure as for the integral images (figure 2.3) can be applied to calculate the contribution of a rectangle to the histogram for channel b. Now, calculating $H(x, y, b)$ for all the histogram channels, the total histogram of oriented gradients for a rectangle can be found by looking up the contribution of the same rectangle to all the channels/orientations.

2.2 Classifiers

The task of a classifier is to determine the most likely class of an object/a subregion of an image. The classification is based on already known instances of each class referred to as the *training set* [46]. i.e a classifier is *trained* a priori to the actual classification. Now, classification, as previously mentioned, may correspond to checking whether an object/subregion of an image is a chair or not (two-class classification). It can also be more complex, e.g *multi-class classification* refers to the case where a classifier is used to determined not only if an object is present, but also distinguishing between different types of objects. In this thesis the focus is on creating classifiers for *two-class classification*. This implies that the training sets consists of only two classes, i.e example images where the object we want to detect is present (*positive examples* or positives for short) and example images where the designated object is *not* present (*negative examples* or negatives for short).

The first step to create a classifier is to use a feature representation in order to describe the training set. Once this is done, there are two ways to proceed. One can either apply *supervised training*, which corresponds to the case where the class of each instance in the training set is made available to the classifier when it is trained. Alternatively, one can use *unsupervised training*, which, as the name implies, is the case where the class of the instances in the training set is *not* given to the classifier during training. Techniques using supervised training has proven to result in fast and accurate classifiers, but later research has shown that unsupervised training can be used in conjunction with supervised training to get enhanced detection results [30]. This, however, is out of the scope of the present thesis. Now, once a classifier is trained on a training set, it can be used to classify objects of interest found in an image. The following section gives a review of two classifiers and takes a brief look at how they can be trained/created for two-class classification using supervised training.

2.2.1 Support Vector Machine

In short, a *Support Vector Machine* (SVM) is a machine learning algorithm that takes labeled training data (positives and negatives) as input. It then proceeds to cluster the data into two classes by finding the maximum marginal hyperplane that separates one class from the other [9]. The margin of the hyperplane is defined as the distance (in feature space) from the hyperplane to the nearest training data point, and the data points that lie on this boundary are referred to as *support vectors* (hence the name). Now, the hyperplane that maximizes the margin, while still separating the positives from the negatives, is said to be *the optimal* hyperplane [23]. An optimal hyperplane in a 2-dimensional feature space can be seen in figure 2.7.

To compute the optimal hyperplane the following definition of a hyperplane is introduced

$$f(\mathbf{x}) = \boldsymbol{\beta_0} + \boldsymbol{\beta}^{\boldsymbol{T}}\mathbf{x} \tag{2.8}$$

where $\boldsymbol{\beta_0}$ is the bias vector, $\boldsymbol{\beta}$ is a weight vector and $\mathbf{x}$ is a vector in the feature space. Notice that the optimal hyperplane can actually be represented in an infinite number of ways. Mathematically this can be done by scaling $\boldsymbol{\beta}$ and $\boldsymbol{\beta_0}$, hence a representation has to be chosen. The one chosen is [23]

$$|\boldsymbol{\beta_0} + \boldsymbol{\beta}^{\boldsymbol{T}}\boldsymbol{\nu}| = 1 \tag{2.9}$$

Where $\boldsymbol{\nu}$ are the training examples closest to the hyperplane in feature space (the support vectors). Equation 2.9 is known as the *canonical hyperplane* [23].

Now, it can be shown [23] that the distance d from a point $\boldsymbol{\nu}$ to the hyperplane $f(\boldsymbol{\nu})$ can be expressed as

$$d = \frac{|\boldsymbol{\beta_0} + \boldsymbol{\beta}^{\boldsymbol{T}}\boldsymbol{\nu}|}{\|\boldsymbol{\beta}\|} \tag{2.10}$$

Furthermore, combining equations 2.9 and 2.10 we can express d in the following way

$$d = \frac{1}{\|\boldsymbol{\beta}\|} \tag{2.11}$$

and from figure 2.7 it can be seen that the total margin M in this case will be equal to

$$M = 2d = \frac{2}{\|\boldsymbol{\beta}\|}$$

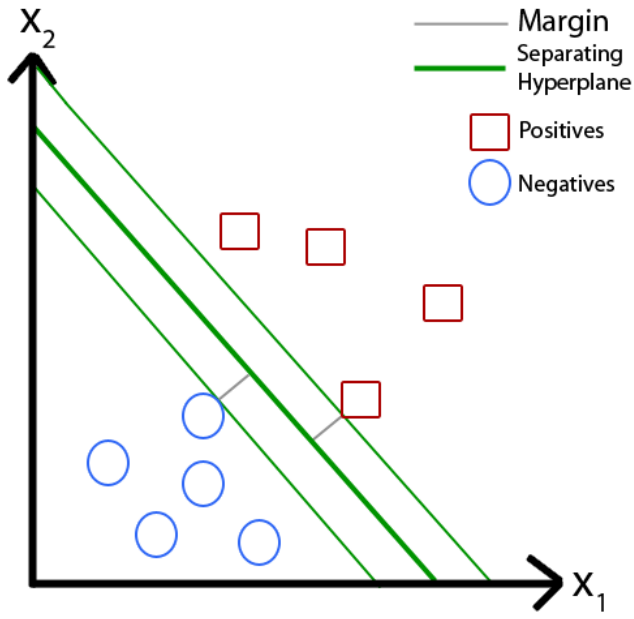

Figure 2.7: A figure showing the separating hyperplane and its relation to other class instances in a two dimensional feature space. It is seen that both margins are equal, hence the hyperplane is placed halfway between the two closest instances of the two classes.

Finally, remembering that the goal of the SVM algorithm is to maximize the margin (M), it is readily seen that this can be expressed as the following optimization problem

$$\begin{aligned} &\min_{\boldsymbol{\beta},\boldsymbol{\beta_0}} L(\boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{\beta}\|^2 \\ &\qquad subject\ to \\ &\mathbf{y_i}(\boldsymbol{\beta}^{\boldsymbol{T}}\boldsymbol{\nu}_i + \boldsymbol{\beta_0}) \geq 1 \quad \forall i \end{aligned} \tag{2.12}$$

Where $\mathbf{y_i} \in \{-1, 1\}$ indicates what class the point $\boldsymbol{\nu}$ belongs to (hence also which side of the hyperplane the data point lies in). This minimization problem can be understood as minimizing $\|\boldsymbol{\beta}\|$ (hence, maximizing M), while maintaining the *canonical hyperplane* (the restrictions). Equations 2.12 represent a *quadratic programming problem* [23], and is a well known optimization problem within optimization theory.

Now, in the cases where there is no solution to this problem (e.g in the case of mislabeled examples), some slack can be imposed on the restrictions. Furthermore, if there is no linear hyperplane that can solve this problem, a kernel trick can be used on the training data. The kernel trick, first proposed by Aizerman et al. [4], transforms the training data to a higher dimensional space which is likely to be linearly separable. The above mentioned optimization problem is then solved in this higher dimensional space, before the inverse of the kernel is applied to find the solution in the original feature space. This is explained in figure 2.8. The reader is referred to [23] for a more in-depth-look on the support vector machine and the non linear case. It should be noted that the weights of $\boldsymbol{\beta}$ satisfying equation 2.12 will represent a linear combination of the support vectors.

Since the SVM classifiers discriminate between data in feature space, the success of such a classifier is very dependent on the feature representation of the test set. It should be rich (large number of dimensions), and in addition cluster similar data. In such regard, the HOG representation covered in section 2.1.2 has proven useful [38] [14]. Classification with a SVM classifier can be done by computing the feature descriptor of a region of interest, and then take the distance from this point to the hyperplane given by the SVM classifier. Depending on the value of this distance (negative or positive), the region of interest can be said to belong to a specific class.

The Haar-features are represented with only a scalar value (1 dimension), hence it is not very useful for creating a SVM classifier. For Haar-features to be effective a different approach is preferred.

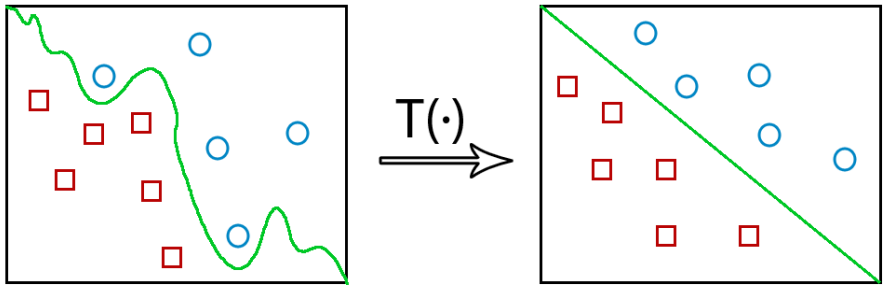

Figure 2.8: Illustration of how applying a kernel $T(\bullet)$ to the training set can transform a nonlinear optimization problem to a linear one. The red squares and the blue circles are the positive and negative training examples, respectively. The green line is the hyperplane separating the two classes from each other.

2.2.2 Boosted Cascade Classifier

A *cascade* classifier consist of several simpler classifiers, which subsequently are applied to a region of interest. That is, each classifier is applied subsequently until a candidate region of interest is either rejected or it passes through all the classifiers [41]. Once a region of interest passes a classifier, it is said that it has passed a *stage*. This concept is illustrated in figure 2.9. Now, the word *boosted* refers to the fact that each classifier (stage) is complex in the way that it is constructed. To be more specific, this means that a weighted voting technique (*boosting technique*) is used to create the classifier at a given stage n. There exists several boosting techniques, but the one used in this thesis is an algorithm called AdaBoost (Adaptive Boost) proposed by [41]. AdaBoost is a boosting technique that, at each stage n, seeks to find the most critical features to evaluate in a region of interest. These features are the most critical ones in the sense that they are, combined, the most effective features with regard to classifying a region as non-object/object. It should be emphasized that the performance of a boosted cascade classifier is greatly influenced by its ability to reject regions of interest which do not contain an instance of the object we are looking for at an early stage. This will greatly reduce the total computation power required for object detection, as each stage of the classifier is fairly simple and does not require much computation. Hence, a good boosted cascade classifier should find the features that are most characteristic for the object we are searching for, and let these features make up the earlier stages of the cascade

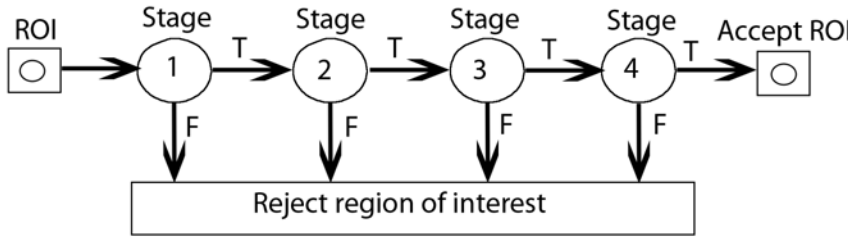

Figure 2.9: A region of interest must pass through *all* stages (in this case 4) to be accepted, i.e classified as an instance of the type of object the classifier is trained to recognize. If any of the 4 stages reject the region of interest, further processing is dropped. ROI = Region of Interest

classifier.

To create a boosted cascade classifier the first thing that has to be done is to find a suitable type of weak classifiers that can be used to make up a classifier at each stage n. A weak classifier is here defined to be a classifier which is restricted to using only a single feature [41], and an example of such a classifier is $h_j(\mathbf{x})$ as described in section 2.1.1. Using the AdaBoost algorithm to construct a boosted classifier consisting of T features was derived by Viola and Jones in [41], and is summarized as follows:

- Find example images (x_1, y_1),...,(x_n, y_n) where $y_i = 0, 1$ for negative and positive examples respectively
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{2}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the numbers of negatives and positives respectively.
- For `t = 1,...,T`
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1} w_{t,j}}$$
  so that w_t is a probability distribution
  2. For each feature, j, train a classifier h_j which is restricted to using a

single feature. The error is evaluated with respect to w_t, i.e

$$\epsilon_j = \sum_i^n w_i |h_j(x_i) - y_i|$$

3. Choose the classifier, h_t, with the lowest error ϵ_t
4. Update the weights:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & if \quad \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases} \tag{2.13}$$

where $\alpha_t = log\frac{1}{\beta_t}$.

An intuitive explanation of the AdaBoost algorithm is seen in figure 2.10. The main observation is that the AdaBoost algorithm systematically selects the features that are best suited for classification (point 2 and 3). Another important aspect of this algorithm is the procedure which is applied in step 4. In this step, the weight on the wrongly classified examples (either false positive or false negative) is increased, making these examples more important than the other already correctly classified objects. By doing so, the weak classifier that the AdaBoost algorithm selects next, will be the one that is able to correctly classify those object that the previously weak classifiers did not classify properly. The main idea is that the cascade of weak classifiers will form a strong classifier by supplementing each other, i.e where one classifier fails, others will succeed. The strong classifier $h(x)$ is referred to as a *boosted classifier*. It should be noted that the requirement for this strong classifier to classify a region of interest as a positive example of an object, is that at least a weighted half of the weak classifiers find their corresponding feature within the region of interest. It is a weighted half because the weak classifiers with low error rate count more towards the final decision.

Now, a boosted *cascade* classifier can be created by combining several boosted classifiers. This is done by training each stage of the cascade classifier with the AdaBoost algorithm. An acceptable false positive rate, and an acceptable target detection rate is set for both each stage n and the cascade classifier in total. Now,

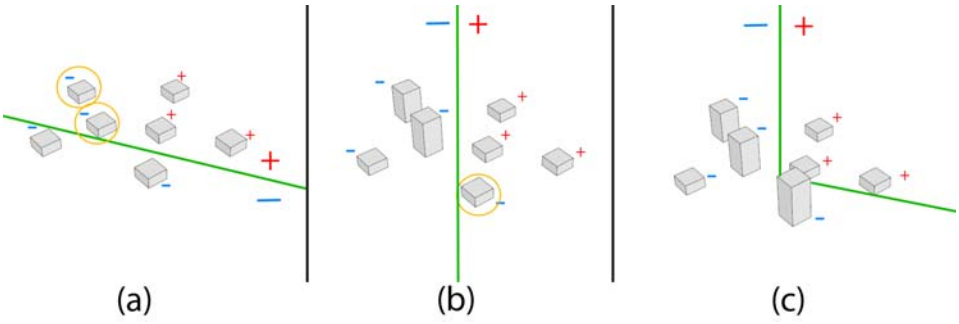

Figure 2.10: Visualization of the AdaBoost algorithm. Each block corresponds to an example in the training data. Blocks with + represent positives, and blocks with − represents negatives. In (a), a classifier is shown represented with a green line. It separates the training data into a positive and a negative side (marked by a big + and −) . The false negatives and the false positives are marked with a yellow circle. (b) shows the next step, where the weight (represented by the height of the block) on the wrongly classified examples has increased. (c) shows how two classifiers can be used to correctly classify all examples in the training set.

each stage is trained by adding features until the target detection and false positives rates are met for that specific stage. Furthermore, stages are added until the *overall* target for false positive and detection rate is met. In this regard, the boosted cascade classifier is very easy to create with good control of the tradeoff between computational cost and accuracy during detection. As previously mentioned it is preferred that the first stages of the cascade classifier consists of few, but effective, weak classifiers.

A boosted cascade classifier using Haar features to create the weak classifiers has proven to be a very powerful combination, and is known to be a very fast and reliable way to perform object detection [43] [20].

Chapter 3

Object Tracking

Object tracking in computer vision can be defined as the problem of estimating the trajectory of an object in the image plane (2 dimensional) as it moves around a 3 dimensional scene [45]. This is done by creating a tracker that assigns consistent labels to tracked objects in every frame of a video. There exists many solutions to this problem, but there are also many factors that makes the tracking of objects a very challenging task. For example complex object motions, partial and/or full object occlusions and real-time processing requirements are all factors that will impact which tracking algorithm that is suited for an application. Furthermore, tracking different objects will often need different tracking algorithms to achieve proper tracking. For example, in an UAV, tracking moving cars is different from tracking humans. A common approach to track fast moving objects from UAVs (like moving cars) is a *differential motion analysis* [7, 37]. This is an approach where the egomotion of the UAV is estimated, so that subsequent frames in a video sequence can be subtracted from each other. This will result in that only the moving objects in the image will remain in the final result, hence they become easy to detect. However, it should be obvious that the same approach can not be used to detect and track humans from an UAV, as the movement (if any) of humans seen from a high altitude often will appear stationary between subsequent frames.

In the present thesis, a tracking algorithm that is able to detect and track both stationary *and* moving objects is preferable. Furthermore, it should be able to track multiple objects, while keeping the computational cost of the tracker algo-

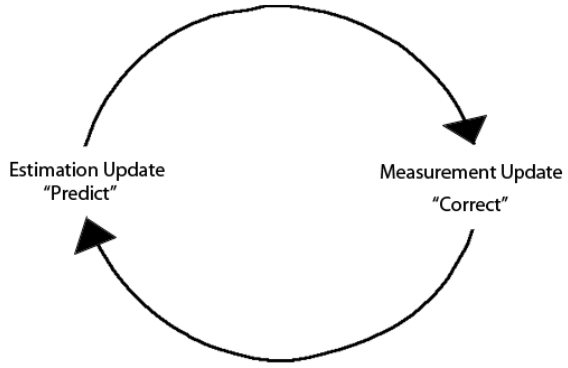

Figure 3.1: The tracking cycle for an estimate-and-measure tracker. The estimation update predicts the position of an object ahead in time. The measurement update corrects this prediction based on the measurement of the actual position of the object.

rithm sufficiently small. This is because the tracking algorithm should be able to run on the on-board UAV hardware in real-time. Considering the limited computational power available, it should stand to reason that the performance of the tracker may have to suffer in favor of simple computation, at least compared to current state-of-the-art trackers. One type of tracking algorithms that are known to fulfill many of these criteria are the ones that are based on an *estimate-and-measure* approach [45]. This means that the tracking algorithm estimates the position of an object into the future using measurements available in the present. This step is referred to as the prediction step. Furthermore, the estimate is updated/corrected by doing measurements of the position of the object, which is referred to as the correction step. This cycle is illustrated in figure 3.1. The following chapter is a review of the tracking problem based on the estimate-and-measure approach, with the goal of arriving at a multi object tracker for application in on-board hardware in UAVs.

3.1 Estimation

In object tracking, estimation is the concept of attempting to predict the location of objects at time step $t+1$ using available measurements at the time step t and some stochastic model of the movement of the object. The first step in order to arrive at such an estimator is to determine the probability distribution of the measured data. In the case of object tracking, this is done by creating a *motion model* for the tracked objects. Next, an estimator known to estimate said motion model efficiently can be developed (or applied), which in turn will allow the estimator to estimate future positions of the tracked objects.

This section will begin with a review of a common motion model for object tracking in computer vision. Following that, an estimator that is known to estimate this motion model efficiently is introduced, before the section ends with a look at how measurements can be used to update the parameters of the estimator.

3.1.1 Motion Model

To track objects using an *estimate and measure*-approach, a motion model for the objects we want to track is needed. A motion model can be derived by assuming that the projections of the objects on the image plane will move in a certain way. The first step to find such a model is to realize that when an object moves around in the real world, *on the image plane* this will be projected as an two dimensional displacement. This is illustrated in figure 3.2. Now, assuming that the motion of the object is fairly smooth (no abrupt changes), motion of an object on the image plane can be modeled as a two dimensional linear motion. Accounting for small variations in the velocity, a discrete time linear motion model might be described by the following equations [13]:

$$\begin{aligned} x_{k+1} &= x_k + \Delta T v_x + w_k \\ y_{k+1} &= y_k + \Delta T v_y + z_k \end{aligned} \tag{3.1}$$

where x_k, y_k is the position of the object in the image at time step k, and v_x, v_y is its velocity. ΔT is the time between each time step (i.e $t(k+1) - t(k) = \Delta T$) and w_k and z_k are gaussian white noise variables accounting for small changes in the velocity. Let us also assume that we want to measure the location of the object in the image plane at time step k, and that the measurement sometimes may be a bit

inaccurate. This can be described in the following way

$$\begin{aligned} y_{1_k} &= x_k + v_k \\ y_{2_k} &= y_k + q_k \end{aligned} \tag{3.2}$$

Here, y_{1_k} refers to the measurement of x_k, and y_{2_k} the measurement of y_k. v_k and q_k is also here assumed to be gaussian white noise, now describing the measurement inaccuracy. Setting $\Delta T = 1$ (without loss of generality), equations 3.1 and 3.2 can be written in the following compact state space form [13]

$$\begin{aligned} \mathbf{x}_{k+1} &= A\mathbf{x}_k + B\mathbf{u}_k + \mathbf{w}_k \\ \mathbf{y}_k &= C\mathbf{x}_k + \mathbf{v}_k \end{aligned} \tag{3.3}$$

Where we have that

- A, B and C are matrices
- k is the time index
- $\mathbf{x}_k$ is the state vector at time step k
- $\mathbf{u}_k$ is the input to the system at time step k
- $\mathbf{y}_k$ is the measurement at time k
- $\mathbf{w}_k$ and $\mathbf{v}_k$ are the *white noise vectors* at time step k, where $\mathbf{w}$ is the process noise and $\mathbf{v}$ is the measurement noise

For the two dimensional motion described in equation 3.1, the state vector $\mathbf{x}_k$, the measurement $\mathbf{y}_k$ and the input to the system $\mathbf{u}_k$ are equal to

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ v_x \\ v_y \end{bmatrix} \quad \mathbf{u}_k = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{y}_k = \begin{bmatrix} y_{1_k} \\ y_{2_k} \end{bmatrix} \tag{3.4}$$

Furthermore, the matrices A, B and C are equal to

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.5}$$

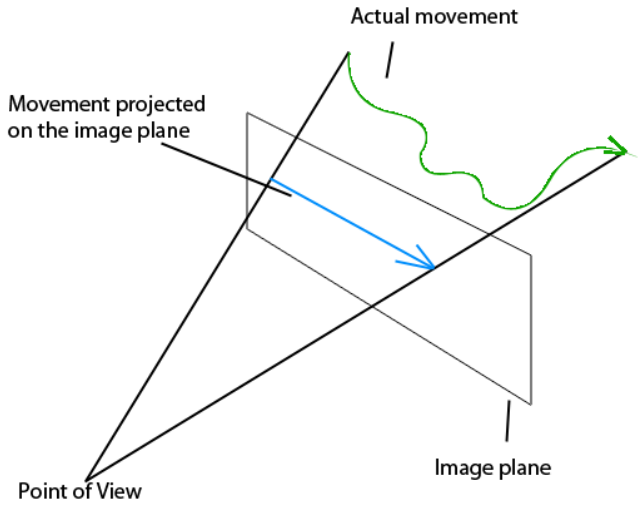

Figure 3.2: This figure shows how an object moving in a complex way can appear to move linearly on the image plane, i.e through the eyes of a camera.

where x, y is the object location in the image, and v_x, v_y is the velocity of the object in the x and y direction respectively.

Now, when tracking is performed from a moving UAV, this will also have an affect on the frame to frame displacement of the tracked objects. Let us, for the simplicity, assume that the UAV has constant velocity and constant attitude. Let us also assume that the UAV will not make any abrupt turns. In this case, movement of the UAV will affect the perceived motion of the tracked objects simply by an offset in the velocity of the objects. This implies that the two dimensional linear motion model still can be effective in the process of tracking objects.

Even though these assumptions are simplifying the problem, the most important characteristic of a motion model is that it should be *accurate enough*. This is a rather vague criteria, but it refers to the fact that the motion model can be simplified, if we compensate for this by making observations. This is the step where the estimator is very powerful. Finally, it should be noted that due to the limited computer power available in-flight, a simple motion model is preferred.

3.1.2 The Kalman Filter

Now that we have acquired a motion model, the next step is to develop or apply an estimator that is known to estimate such linear motion models accurately. In this regard, the Kalman filter is a very common and effective estimator for predicting the state of a linear motion model. That is, in many cases the actual states $\mathbf{x}_k$ are unavailable/unknown, and the Kalman filter can be used to predict the states $\mathbf{x}_k$ based on measurements of the type $\mathbf{y}_k$. The Kalman filter can also be used to predict $\mathbf{x}_{k+1}$ (prediction into the future) based on all previous measurements up to and including $\mathbf{y}_k$.

It could be tempting to use the measurements $\mathbf{y}_k$ directly to estimate the state of the system. However, because the measurements are corrupted by the noise $\mathbf{v}_k$, this is not an efficient approach. Hence, the Kalman filter does not only use the measurement $\mathbf{y}_k$, but also uses the information that is contained in the state equation. More precisely, the Kalman filter is a set of equations that provides a recursive solution to the following least-squares problem [24]

$$\min_{\hat{\mathbf{x}}_{k|k}} E\big[|\mathbf{x_k} - \hat{\mathbf{x}}_{\mathbf{k}|\mathbf{k}}|^2\big] \tag{3.6}$$

where $\hat{\mathbf{x}}_{k|k}$ is the estimate of the state $\mathbf{x}_k$ given measurement up to and including time step k. This is also referred to as the *a posteriori*[1] state estimate. The reader is referred to [24] for a detailed derivation of the Kalman filter, but it should be noted that it is constructed in such a way that each measurement casts a weighted vote towards the estimation of the linear model (equation 3.3). That is, the certainty of a measurement is taken into account when the Kalman filter is updated. If a measurement is likely to be an anomaly or a very noisy measurement, the Kalman filter weights this measurement lower than a measurement that fits the motion model better.

Now, solving equation 3.6 yields the following set of recursive equations [24].

$$\begin{aligned} K_k &= P_k C^T (C P_k C^T + R)^{-1} \\ \hat{\mathbf{x}}_{k+1} &= (A\hat{\mathbf{x}}_k + B\mathbf{u}_k) + K_k(\mathbf{y}_k - C\hat{\mathbf{x}}_k) \\ P_{k+1} &= A(I - K_k C)P_k A^T + Q \end{aligned} \tag{3.7}$$

- $\hat{\mathbf{x}}_k$ is the estimate of $\mathbf{x}_k$

[1] A posteriori is Latin for "from the later", and refers to the fact that $\hat{\mathbf{x}}_{k|k}$ is an estimate of the states done *after* measuring the system output $\mathbf{y}_k$ at time step k

- K_k is a matrix called the Kalman gain
- P_k is a matrix called the estimation error covariance, and is defined as

$$P_k = E([\mathbf{x}_k - \hat{\mathbf{x}}_k][\mathbf{x}_k - \hat{\mathbf{x}}_k]^T)$$

- Q is the covariance matrix of the process noise $\mathbf{w}_k$, and R is the covariance matrix of the measurement noise $\mathbf{v}_k$. We have

$$Q = cov(\mathbf{w}_k, \mathbf{w}_j) \qquad R = cov(\mathbf{v}_k, \mathbf{v}_j)$$

- A, B, C, $\mathbf{u}_k$ and $\mathbf{y}_k$ are as defined in equation 3.3
- The -1 superscript indicates matrix inversion
- The T superscript indicates matrix transposition
- I is the identity matrix

Another representation of the Kalman filter which is more useful in implementation is a representation where the calculations are divided into one *predicting* and one *updating* phase. This is more in line with the estimate-and-measure approach to tracking, and make the Kalman filter easy to manage. The predict phase produces a state estimate that is referred to as the *a priori*[2] state estimate. Mathematically, it is denoted $\hat{\mathbf{x}}_{k|k-1}$ as it is an estimate of the state vector based only on previous measurements. In the update phase, we have that the *a priori* prediction is combined with the new measurement information to correct the state estimates from the predict phase. Equation wise, the predict phase can be represented by [24]

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= A\hat{\mathbf{x}}_{k-1|k-1} + B\mathbf{u}_{k-1} \\ P_{k|k-1} &= AP_{k-1|k-1}A^T + Q_k \end{aligned} \tag{3.8}$$

and for the update phase, the equations are

$$\begin{aligned} K_k &= P_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + K_k(\mathbf{y}_k - C\hat{\mathbf{x}}_{k|k-1}) \\ P_{k|k} &= (I - K_kC)P_{k|k-1} \end{aligned} \tag{3.9}$$

Note that equations 3.8 and 3.9 combined are equal to equation 3.6.

[2] A priori is Latin for "from earlier", and refers in this case to the fact that the Kalman filter tries to estimate the state vector using only previously measured data ($\mathbf{y}_k$ is not yet available).

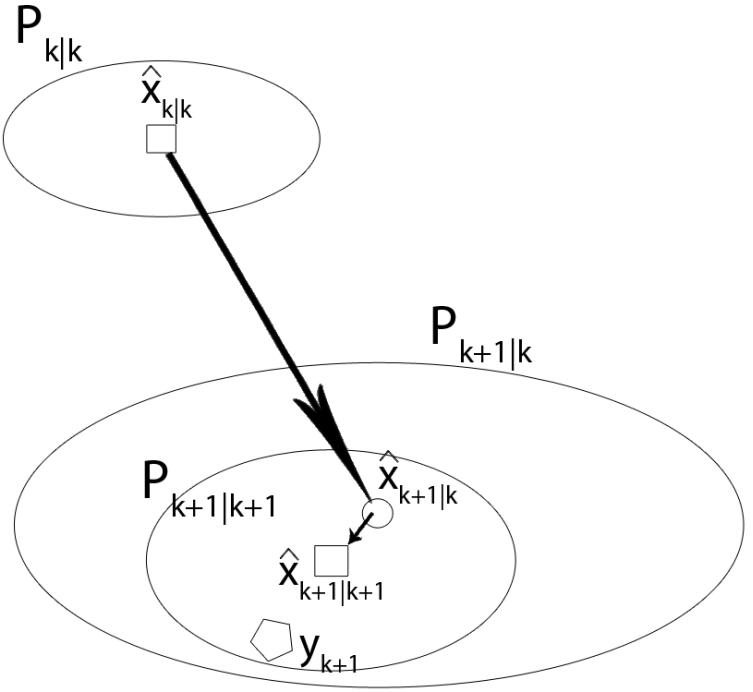

Figure 3.3: The position and velocity of the object at time $k+1$ is predicted (the smallest circle) using measurements available at time k. Then, an observation is made at y_{k+1} (the pentagon), and the prediction is corrected using this information ($\hat{\mathbf{x}}_{k+1|k+1}$). Note how the estimation error covariance matrix P increases at the prediction, and then decreases when new measurements are available.

3.1.3 Tracking With the Kalman Filter

After having established how general estimate-and-measure trackers work, and also having found an estimator that can be used in such algorithms, the big question still remains. How can such estimators be used to track objects? The tracking method can be summarized in the following steps [5]:

1. **Find object**
   Before tracking of the object can begin, the image location of the object has to be found. This can be done by applying an object recognition algorithm (e.g one of the techniques reviewed in chapter 2) on *the whole image*. Furthermore, the estimation error covariance is initially set high, as the state estimate of the motion model is still uncertain. The initial state estimate is based on the observed position of the object. In summary

   $$\hat{\mathbf{x}}_{0|0} = \begin{bmatrix} x \\ y \\ 0 \\ 0 \end{bmatrix} \qquad P_{0|0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   Where (x, y) is the observed position of the object.

2. **Estimate**
   In this step, the estimator (e.g Kalman filter) is used to predict the image position of the object. This position is then used as the search center for the object. Note that the estimate used is the *a priori* $\hat{\mathbf{x}}_{k|k-1}$.

3. **Measure**
   After locating the object within the search area estimated in step 2, the measured location of the object is used to correct the state estimation, i.e calculating $\hat{\mathbf{x}}_{k|k}$. After this is done, steps 2 and 3 are repeated on the next image frame.

Steps 2 and 3 are illustrated in figure 3.4.

Now, when this approach is used to track objects, there are some considerations that have to be made. Specifically, care has to be taken with regard to the following points

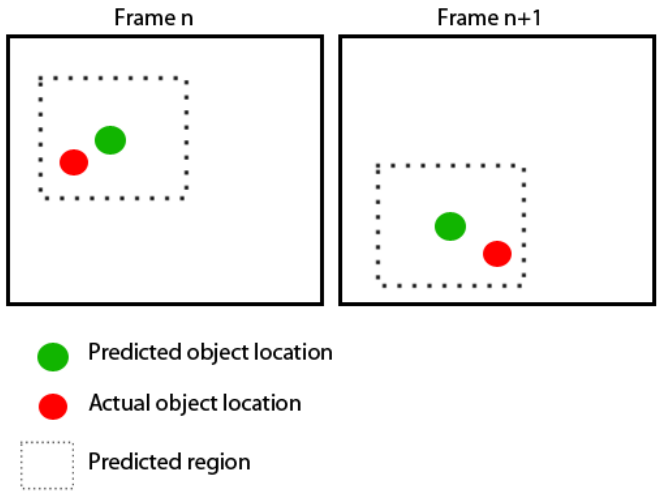

Figure 3.4: An illustration of how the Kalman filter is used to predict the location of an object in each frame, and how the search for said object can be limited to a subregion of the image frame.

Scaling of the predicted region
To be certain that the tracked object will be located within the predicted region, the predicted region should be scaled both according to the size of the object, *and* the certainty of the prediction. A measure of the certainty of the estimate in the Kalman filter is the estimation error covariance matrix P_k. Hence, scaling the predicted region based on P_k may be a good idea. The region should also be scaled at some percentage of the size of the object, and not some constant value.

No measurement available
In some image frames the detection algorithm may not be able to localize the object. This can for example be the result of the fact that the object is partly or fully occluded. A reasonable way to solve this problem is to use the predicted location of the object as the measurement, i.e the Kalman filter supplies itself with a measurement. A criterion for this to work properly is that the estimated motion model should be accurate.

Choice of the Q and R matrices
The choice for the noise covariance matrices Q and R will greatly impact the tracking abilities of the estimator. There has been some research on the subject of choosing Q and R based on empirical data, but good performance can also be achieved by trying different variations.

The Kalman filter can also be used to track multiple objects, simply by creating a new instance of a Kalman filter for each new object that is observed. New objects can be located by searching the whole image frame from time to time. A new Kalman filter is then created for objects that do not have a high correlation with the position of an object that is already being tracked. How this is done in practice will be described in section 3.2.2.

3.2 Measurements

As already mentioned, the problem of object tracking can be defined as that of estimating the movement of an object in the image plane. As described in the previous section, some tracking algorithms seek to achieve this by creating a motion model for the tracked objects. Now, estimating the states of such motion models relies on measurements, which in turn raises the question of what type of measurements,

and of what form these measurements should be. There are as many answers to these questions as there are trackers, and there is no definite best answer [45]. In other words, different trackers will require different types of measurements. However, the best measurement for a given tracker algorithm, is the measurement that *most effectively* supply the tracker with information that can be used to correct or enhance the estimated object motion. Furthermore, the frequency and quality of the measurements has to be decided. In many real-time tracking applications a good balance between quantity and quality is very important. That is, over a given time period it may be better to have two inaccurate measurements, than to have one really accurate one.

The following section features a review of techniques that can be applied to an image to find accurate measurements in an efficient way, followed by a simple approach to match measurements to the correct Kalman filter in the case of multiple object tracking.

3.2.1 Finding Measurements

Finding measurements can be a very time consuming task, especially if it is not done efficiently. The easiest, but also the most computational heavy approach, is simply to search the whole image for the tracked object. However, this is an approach that can take up to seconds even on powerful desktop computers. Hence, with the computational power available on an UAV, it may be preferred to apply filtration methods that narrows the search to a few, but important, regions in the image. A method to locate regions of interest is listed below.

Normalization

Even though some object detectors are invariant to changes in lighting and intensity, it is favorable to find some sort of common ground that every frame in a video sequence can be evaluated from. This can, to some degree, be obtained by normalizing each frame in a consistent way. Assuming $i(x, y, t)$ is the image captured at time t, the following equation can be applied to normalize $i(x, y, t)$

$$i'(x, y, t) = \frac{i(x, y, t) \times \gamma}{\bar{i}(t)} \tag{3.10}$$

where $i'(x,y,t)$ is the normalized image, $\bar{i}(t)$ is the mean intensity value of the image i and γ is a scaling factor [11]. Note that when $\bar{i}(t) = \gamma$, we have that $i'(x,y,t) = i(x,y,t)$. The normalization is done such that every image will have similar intensity values, which is a good feature to have in the search for relative light and dark regions.

Segmentation by Thresholding

As mentioned, searching the whole image in every frame is an inefficient solution to the tracking problem, hence we want to divide the image into different regions. Preferably, the image should be divided into interesting and non-interesting regions. This process is called *image segmentation*. Segmentation can be achieved in many ways, but an effective technique on gray scale images is referred to as *thresholding*. Thresholding can be used to identify regions that are either lighter or darker than its environment. This is done in the following way [39]

$$i(x,y) = \begin{cases} maxValue & if \quad i(x,y) > \theta_T \\ 0 & otherwise \end{cases} \tag{3.11}$$

where θ_T is a threshold value between $[minValue, maxValue]$, which is the minimum and the maximum value of the image $i(x,y)$. The process of thresholding is often referred to as *binarizing* [11] the image, because after the thresholding is done, the image can be represented using only two values (0 and $maxValue$). The result of normalizing and thresholding an image can be seen in figure 3.5.

In some situations, the intensity value of the regions we are interested in locating is somewhere *in between* two different intensity values. In this case, two-tailed thresholding can be used. This is formally defined as

$$i(x,y) = \begin{cases} maxValue & if \quad \theta_{T_1} \leq i(x,y) \leq \theta_{T_2} \\ 0 & otherwise \end{cases} \tag{3.12}$$

Now, after applying thresholding to an image, it will contain a number of connected regions called *blobs*. This image can be processed further to remove blobs that do not have a shape similar to the tracked object.

Figure 3.5: (a) is the original image. (b) shows a normalized version, which has resulted in a (slightly) larger contrast. (c) is the thresholded image with thresholding value (θ_T) at 150 (max is 255).

Erosion and Dilatation

After applying the thresholding technique on an image, several non-interesting regions may still be a part of the image. This can happen if either some parts of the background or other objects than the one being tracked, has similar intensity values as the tracked object. Then, to favor objects of one certain shape, *morphological operations* can be applied. Morphological operations are operations that process images based on shapes. This is done by applying a *structuring element* to an image, generating a new image that is sensitive to the shape of the structuring element. There are two types of basic morphological operations, i.e erosion and dilatation [39].

Dilation is an operation where the output pixel is the *maximum* value of all the pixels in the neighborhood of the input pixel, effectively *adding* pixels to the boundary of a blob. Exactly what neighborhood this is, is defined by the structuring element. Erosion is the opposite, i.e the output pixel is the *minimum* value of all the pixels in the neighborhood of the input pixel, effectively *removing* pixels from the boundary of a blob. Now, erosion and dilatation become especially powerful when they are applied in a specific order. Specifically, erosion followed by dilatation is referred to as *closing*, and dilatation followed by erosion is referred to as *opening*. Opening will remove small blobs present in the thresholded image, while a closing will close the gaps between blobs that lie next to each other. It is the *opening* operation that can be used to find specific shapes in an image, i.e removing blobs that do not have a similar shape to that of the structuring element. The result of applying a *morphological opening* with a structuring element formed as a rectangle

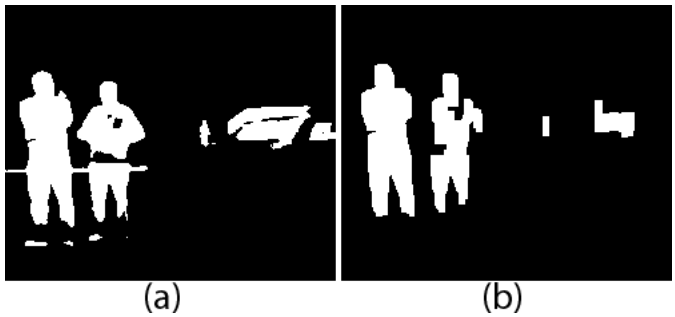

Figure 3.6: (a) is the original thresholded image, and (b) is the image after a morphological opening with a structuring element formed as a vertical rectangle (20x5 pixels).

on a thresholded image is shown in figure 3.6.

Extracting Regions of Interest

By applying a combination of the above mentioned techniques, it is possible to isolate promising features in an image. The regions in the image where these features are located are referred to as *regions of interest*. Now, these are the regions that we want to extract from the original image and examine more closely. A common way to do this is to use *connected-component labeling* on the thresholded image [39]. The result of a connected-components algorithm is that each blob (collection of bright pixels) in a segmented image is appointed its own unique label. There exists different connected-components algorithms, and they differ mainly in which pixels are considered to be neighboring pixels. *4-connectedness* and *8-connectedness* are the most commonly used algorithms. In 4-connectedness, a pixel is only part of a blob if it has a bright neighboring pixel for at least one of four locations (north, west, south and east). In 8-connectedness the only requirement for a pixel to be part of a blob is that any of its 8 neighbor pixels should also be bright. The result of applying *8-connectedness based connected-component labeling* can be seen in figure 3.7.

Now, the area of each connected component can then be calculated, which in turn allows for the filtration of either too small or too big blobs. This is an effective way to isolate the features that have a size similar to that of the tracked object.

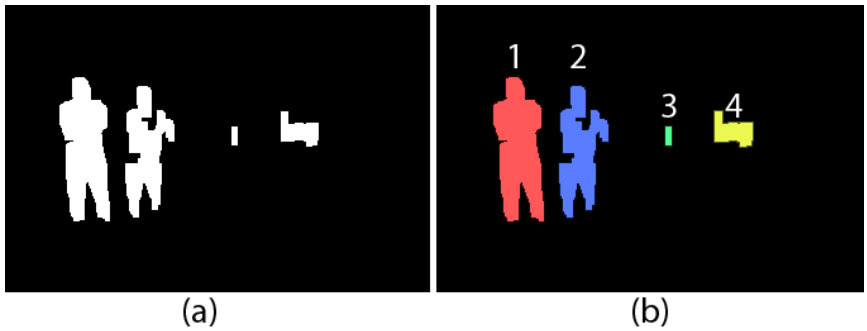

Figure 3.7: The result of applying connected-components labeling is that each blob is appointed its own unique label. In this case, there are 4 blobs present.

Finally, the features that are still present in the image represent regions of interest. The position of the center of these features can then be used as a search center for the tracked object in the original image. The size of the region around the search center that is extracted from the original image is scaled by the size of the tracked object.

Measurement Representation

As seen in section 3.1, estimate-and-measure trackers use measurements of the *position* of the tracked object in order to do more accurate predictions. Now, the object recognition algorithms in chapter 2 are only able to tell if a whole region is similar to an object or not, i.e, given a region of interest it is either classified as an object or not an object. In order to use this information to update the Kalman filter, we need to represent the whole region containing the object using only a *single point*. This can be done by assuming that the object will be the center of the region of interest, hence the center of said region will be a good point estimate of the position of the object. This point is then passed to the Kalman filter as a measurement. An alternative method could be to calculate something called the central moment of the object, and by doing so calculating the position of the center of the object. However, because of the limited computational power available, the simple solution (region center) is preferred.

3.2.2 Matching Measurements

The problem of matching the obtained measurements to the correct Kalman filter is a problem of *data association*, and this is known to be a very hard and complex problem. In fact, according to [18], data association may be the biggest source of difficulties in computer vision applications in general. In the case of multiple object tracking, an unknown number of tracked objects is something that complicates the problem. This is because new objects can appear or disappear at any time, anywhere. Furthermore, it has to be determined if a measurement is correct or incorrect, and whether a new Kalman filter should be initialized or an existing one should be updated.

In the present thesis, the requirement of low computational cost is weighted very high. In this regard, the problem of data association can be solved by using a *nearest neighbor standard filter* [36]. The nearest neighbor standard filter (hereby denoted NNSF), is regarded as one of the most straight forward approaches to data association. Given several possible measurements for the position of a target, the correct measurement is assumed to be the measurement "closest" to the predicted position of the tracked target. Mathematically this can be defined as

$$\mathbf{y}_\star(k) = \min_{\mathbf{y}\in Y(k)} D(\mathbf{y}) \tag{3.13}$$

where $\mathbf{y}_\star(k)$ is the measurement that is assumed to be correct at time step k, $Y(k)$ contains all measurements available at time step k, and $D(\mathbf{y})$ is the following operator

$$D(\mathbf{y}) = [\mathbf{y} - \hat{\mathbf{y}}_{k|k-1}]^T S(k)^{-1} [\mathbf{y} - \hat{\mathbf{y}}_{k|k-1}] \tag{3.14}$$

in which $\hat{\mathbf{y}}_{k|k-1}$ is the *a priori* predicted measurement (i.e $\hat{\mathbf{y}}_{k|k-1} = C\hat{\mathbf{x}}_{k|k-1}$) and $S(k)$ is its associated covariance matrix. When NNSF is used in combination with the Kalman filter, $S(k)$ will be equal to $CP_{k|k-1}C^T + R$ [36], where $C, P_{k|k-1}$ and R are defined as in equations 3.3 and 3.7. Notice that the matrix $S(k)$ can be regarded as a scaling matrix, effectively scaling the distance between the estimated state and the measurement according to how certain the estimate is. If it is a certain estimate, this indicates a small R and/or P (yielding a small $S(k)$), which in turn results in a large $S(k)^{-1}$, meaning that the distance $D(\mathbf{y})$ is scaled to be *greater* than it actually is. In the same way, the distance between a measurement and an uncertain estimate is scaled to be *shorter* than it actually is, since such an estimate indicates a large R and/or P.

Now, in the case of *multiple object tracking*, some considerations have to be made.

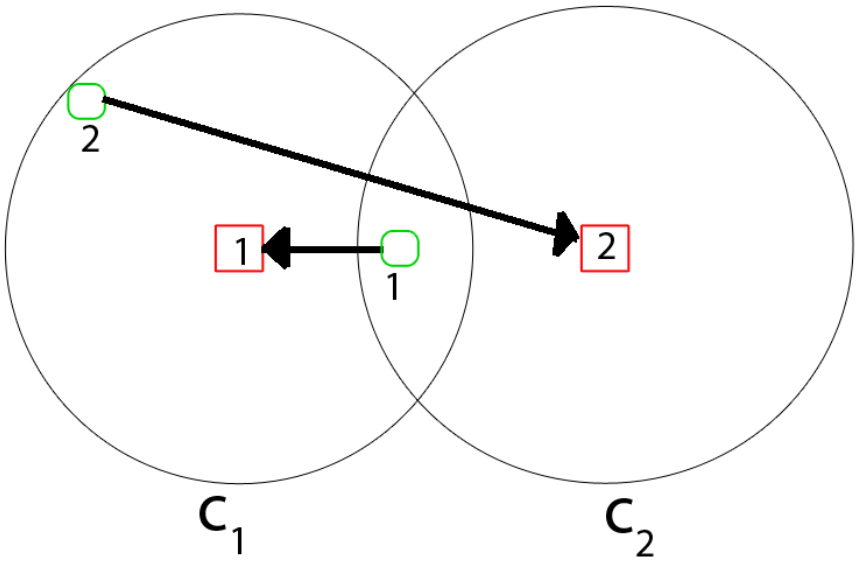

Figure 3.8: In multiple object tracking, the nearest neighbor standard filter is prone to error, as the order of matching measurements can affect the result. Assuming that the green boxes are measurements, and that the red boxes are estimated positions of objects obtained by application of Kalman filters, it is seen how starting with measurement 1 (hence matching it to object 1) leads to a non optimal matching. Since measurement 1 is matched to object 1, the only object measurement 2 can be matched to is object 2. However, this does not equal to the optimal solution (which is measurement 1 matched to object 2, and measurement 2 matched to object 1). The circles C_1 and C_2 are circles indicating the boundary where the position measurements for each object are most likely to be found.

There are some problems with applying the same naïve approach as described in equations 3.13 and 3.14 for several objects. Assume that $Y(k)$ is all available measurements, now originating from n different object sources, and that they should be matched to m different Kalman filters. By simply assigning the measurement closest to the predicted state for each Kalman filter (in a sequential order), the total distance between the Kalman filters and the measurements may very well *not* be the global minimum solution. This is illustrated in figure 3.8. To find the global minimum solution in the case of multiple object tracking, a *global nearest neighbor* (GNN) algorithm can be applied [25]. This is a more computationally heavy approach, but is very similar to the NNSF algorithm. When the global minimum distance is to be found, the following distance matrix A is calculated

$$A = \begin{bmatrix} D_{1,1} & D_{1,2} & \cdots & D_{1,m} \\ D_{2,1} & D_{2,2} & \cdots & D_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ D_{n,1} & D_{n,2} & \cdots & D_{n,m} \end{bmatrix} \tag{3.15}$$

where $D_{i,j}$ is the distance (equation 3.14) from measurement i to the predicted measurement $\hat{\mathbf{y}}_{k|k-1}$ of Kalman filter j. If the distance $D_{i,j}$ exceeds some threshold c_j, $D_{i,j}$ is set to infinity. This is because in the case of

$$D_{i,j} \geq c_j$$

the measurement i is not very likely to be a measurement of the position of the tracked object j. Hence, if all values along a column j are equal to infinity, it is assumed that a measurement for Kalman filter j is not present. To cope with this, column j should be removed from A, and the *predicted object position* should be used as the best available measurement for Kalman filter j. Furthermore, if all values along a row i is equal to infinity, it is assumed that there exists no probable Kalman filter for measurement i. In other words, this is most likely a measurement originating from a new, unknown object. Hence, row i should be removed from the matrix A, and a new Kalman filter should be initialized with measurement i as the initial state.

After calculating A and removing unneeded rows and columns, the data association problem is a matter of finding the combination of distances $D_{i,j}$ that yields the global minimum distance. That is, the combination that assigns *one* measurement to *one* Kalman filter, in a way such that the total distance between all measurements and their assigned Kalman filters is the shortest achievable distance. This is a well studied problem, and can be solved by applying an algorithm called the *Kuhn-Munkres algorithm*. The details of this algorithm can be found in [47],

but in short, it is an algorithm that solves the following optimization problem [25]

$$\begin{aligned} \min(\textit{total distance}) &= \min \sum_{i=1}^{N} \sum_{j=1}^{N} D_{i,j}\chi_{i,j} \\ &\textit{subject to} \\ \sum_{j=1}^{N} \chi_{i,j} &= 1 \quad \forall i \in \{1, N\} \\ \sum_{i=1}^{N} \chi_{i,j} &= 1 \quad \forall j \in \{1, N\} \end{aligned} \tag{3.16}$$

where $D_{i,j}$ is defined as before, $\chi_{i,j} = 1$ if and only if measurement i is assigned to Kalman filter j, and 0 otherwise. The restrictions are the requirement of *one-to-one* mapping, i.e it limits one measurement to one Kalman filter, and one Kalman filter to one measurement. After this optimization problem is solved, the global nearest neighbor solution is found, and the Kalman filters can be updated thereafter.

Chapter 4

The System

Since object detection and tracking are going to be performed on-board the UAV, and not in some external control center, a somewhat complex sensory payload is needed for this task. Ideally, the payload should be able to perform the object detection and tracking in real time, stream video to ground station, log data (such as object locations) and send commands to the autopilot. Furthermore, since the payload is located in an UAV, it is preferable if the electrical power requirement is as low as possible. This is because high power requirement results in a large battery, which in turn results in a high power required to keep the UAV in the air. Now, while taking all of these requirements into consideration, the payload should be able to feature the following modules

- Infrared camera module
- Video camera module
- Navigation module
- CPU module (with interface to the other modules)
- IP radio/communications module
- Battery module

Furthermore, the payload should be as modularized as possible, making it an easy task to change a single component without having to change anything else.

In the following chapter, several different components that can be used in the payload are reviewed, followed by a short description of the software that was used to realize the functionality of the payload, before the chapter ends with the description of two suggested setup configurations. One of these configurations is coined at smaller UAVs, and should be used when the allowed all up weight (gross weight) is small. The other configuration is more advanced and rich with respect to number of features, but is for use in larger UAVs where the allowed weight of the payload is higher.

4.1 Hardware

The general approach in the search for suitable hardware was to first make a broad search of existing components. This was achieved mainly through extensive use of the Internet, but dialogs with firms that specialize within the field of UAVs/robotics were also initiated. After getting a rough overview of available components, a list with requirements for each component was made. This in turn, narrowed down the search to only a few viable candidate components. Following is a review of this process and its outcome.

4.1.1 Single-Board Computer

A single-board computers (SBC for short), is a complete computer integrated on a single circuit board. In other words, the central processing unit (CPU), the memory (RAM), input-output (I/O) modules and other features are seamlessly integrated into one small board. The SBCs are classified by the type of chip that is used to integrate and connect all the different components, which is referred to as the *system on chip* (SOC). Now, the high level of integration of the components in SBCs allows for very compact, light weight and power efficient systems, making them ideal for the use in UAVs. Alternatives such as the Mini-ITX platform ($17cm \times 17cm$ motherboards) is usually more powerful than most SBCs, but the power requirement of such systems is in the range of $30W - 60W$ compared to the usual $5W - 10W$ requirement of a SBC. Furthermore, even though the weight of

Table 4.1: An overview of different ARM based SBCs

ARM SYSTEMS	CPU Speed	RAM	Size	Weight	Power Requirement	Price
CubieBoard	1GHz C-A8	1GB DDR3	$10cm \times 6cm$	50g	5V/2A (10W)	$50
Hackberry Board	1.2GHz C-A8	1GB DDR3	$8.6cm \times 5.4cm$	51g	5V/0.4A (2W)	$65
Nitrogen6X	QC 1GHz C-A9	1GB DDR3	$11.4cm \times 7.6cm$	90g	5V/0.3A (1.5W)	$199
Odroid-U2	QC 1.7Ghz C-A9	2GB DDR2	$6cm \times 6cm$	130g	5V/2A (10W)	$89
Odroid-X2	QC 1.7GHz C-A9	2GB DDR2	$9cm \times 9.5cm$	125g	5V/2A (10W)	$135
Olimex OLinuXino	1GHz C-A8	512MB DDR3	$12cm \times 12cm$	100g	6-16V/0.4A (4.2W)	$60
PandaBoard ES	DC 1.2GHz C-A9	1GB DDR2	$11.4cm \times 10.2cm$	81.5g	5V/1A (5W)	$182

DC = Dual Core QC = Quad Core C-A8/9 = ARM Architecture

Mini-ATX motherboards are similar to some SBCs, most Mini-ATX boards require extra hardware (hence also extra weight) to have the same functionality as SBCs.

In recent years, SBCs have gained a lot of momentum due to their application in mobile devices. The ARM architecture[1] has become especially popular, due to its low cost, high computational power and low power requirements. In other words, within the domain of single-board computers there are many different viable choices for a fast and power efficient computer, and finding the SBC best suited for an application can be hard. In the search for a good SBC, these are the characteristics that were considered most important

- CPU speed (at least a clock speed of 1GHz)
- Sufficient amount of RAM (at least 1GB)
- Low weight (maximum $150g$)
- Small in size (no larger than $15cm \times 15cm$)
- Low power consumption (max $10W$)
- Compatibility with GNU/Linux software

Following these criteria, the search was narrowed down to just a handful of SBCs. A list of the possible choices with their specifications was generated (see table 4.1), and the boards were evaluated. From this list, the CubieBoard, Hackberry Board and the Olimex OLinuXino were excluded almost immediately, as they have *single*

[1] As of 2013, ARM is the most widely used 32-bit instruction set architecture in terms of number of manufactured units

core CPUs. Even though the first two only have weights of $50g$ and $51g$ (respectively), and have very low power consumption, the processing speed is not sufficient to do object detection and recognition. This is especially true in scenarios where the SBC also has to read from/write to file *and* stream video over the network.

The Nitrogen6X is a very interesting board, delivering the most computational power pr Watt with its quad core CPU clocked at $1GHz$. However, to utilize all four cores efficiently, the object detection and tracking algorithms have to be made with multi threading[2] in focus. This is a somewhat complicated task, and since this thesis only seeks to implement a simple tracking algorithm, the algorithm will mainly run in only *one* core at any given time. Hence the Nitrogen6X was also excluded. Of the three SBCs still left on the list, i.e the PandaBoard ES, Odroid-U2 and Odroid-X2, the two latter boards were released to the public in January 2013. At that time, no support for GNU/Linux operating systems had been added, and there was a huge lead time on the boards as there is only one distributor of the Odroid platform, which is located in South Korea. Furthermore, even though both these boards have significantly superior computation power to that of the other SBCs, some of this CPU power would be left unused until the detection and tracking algorithm was tailor made for multi threading. Adding this to the fact that both Odroid-U2 and Odroid-X2 come with a heat sink that weigh around $\sim 100g$, the PandaBoard ES was considered the best choice of all the evaluated SBCs. A schematic and more in-depth info on the PandaBoard ES can be found in appendix C. The PandaBoard ES is relatively expensive (compared to the other alternatives), but it is very light weight, has two CPU cores clocked at $1.2GHz$ and requires little electrical power. As it is not a completely new board, it also has a quite large community resulting in a rich pool of available software and tools.

4.1.2 Infrared Camera

The first crucial decision in the search for an infrared camera (IR camera for short) is whether it should be a *cooled* or *uncooled* camera. Cooling of IR cameras is done to prevent the cameras to be 'blinded' by their own radiation, hence it increases the sensitivity and over all performance of the camera. However, cooling is a power hungry and time consuming task. The cooling mechanism also adds to the

[2]Multi threading is the concept of dividing a problem into smaller parts, where each part can be solved independently. Each part of the problem is then assigned to different CPU cores, effectively decreasing the necessary computational time for a given problem. This is sometimes referred to as *parallel computing*

Table 4.2: Commonly used sub-division scheme for infrared radiation

Division Name	**Abbreviation**	**Wavelength**
Near-infrared	NIR	$0.75 - 1.4\mu m$
Short-wavelength Infrared	SWIR	$1.4 - 3\mu m$
Mid-wavelength Infrared	MWIR	$3 - 8\mu m$
Long-wavelength Infrared	LWIR	$8 - 15\mu m$
Far Infrared	FIR	$15 - 1000\mu m$

total net weight of the camera, which is an important factor in the application of IR cameras in UAVs. Furthermore, cooled IR cameras are in general more costly than the uncooled alternative. Weighting all of these factors, an uncooled IR camera was considered the best solution. The next question was which type of infrared radiation the camera should be sensitive to. As seen in table 4.4, there are different types of infrared radiation. Now, the *long-wavelength* infrared radiation is usually referred to as the *thermal imaging* region, because this is the region in which sensors can obtain a completely passive image of objects with a temperature slightly higher than room temperature (e.g the human body). Sensors operating in this region do no require *any* illumination source such as the sun or the moon, and was considered a good complementation to that of electro-optical (EO) cameras (which is only sensitive to visible light). Hence, the search was narrowed down to a *long-wavelength, uncooled* infrared camera suitable for use in UAVs. To be more specific, the following characteristics were considered the most important

- Uncooled
- Long-wavelength infrared sensitive
- Low weight (maximum $200g$)
- Small in size (less than $10cm \times 10cm \times 10cm$)
- Low power requirement (max $5W$)
- Middle to high resolution (more than 320×240 pixels)

The last point may be a bit misleading, but a resolution equal to $320x240$ pixels is actually considered a middle to high resolution within the field of infrared optics. In fact, the highest resolution of IR cameras open to the public is at $640x512$. Now, the search for IR cameras with the above mentioned characteristics resulted in the

Table 4.3: An overview of different long-wavelength uncooled infrared cameras.

Camera	Resolution	Size (w/o lens)	Power Req	Weight	Sensitivity	Output
A35Sc	320×256	$10.6 \times 4 \times 4.3\,cm^3$	$\leq 2.5W$	$200g$	$\leq 50mK$	Ethernet
Eye-R25	384×288	$5 \times 6.3 \times 6.3cm^3$	$\leq 2.2W$	$\leq 150g$	$\leq 50mK$	Analog (CCIR)
Gobi-640	640×480	$5 \times 5 \times 7.7cm^3$	$2W$	$200g$	$\leq 50mK$	Ethernet
Tau2 640	640×512	$4.5 \times 4.5 \times 3\,cm^3$	$\sim 1.2W$	$72g$	$\leq 50mK$	Analog (BNC)
Tau2 336	336×256	$4.5 \times 4.5 \times 3\,cm^3$	$\leq 1W$	$72g$	$\leq 50mK$	Analog (BNC)
Vumii OFC	$640x480$	$7 \times 5.7 \times 3.9cm^3$	$\sim 2.5W$	$200g$	$\leq 70mK$	Analog (CCIR)

six cameras listed in table 4.3.

Of the six cameras listed, the Vumii OFC was excluded immediately, as it is both the heaviest and the most power hungry camera of the six. Furthermore, even though direct output to Ethernet (A35Sc and Gobi-640) is a desirable feature, the extra weight compared to e.g Tau2 640 is a little too high. The benefit of having a direct output to Ethernet is that it eliminates the need for some other means to convert the analog signal to a digital signal, so that it can be processed by the SBC. However, there exists frame grabbers[3] that weighs less than the weight difference between Tau640 and the Ethernet solutions (Gobi-640 and A35Sc). In appendix F two such frame grabbers are suggested, where one of them connects through USB (EasyCap) and the other through Ethernet (Axis M7001). Hence, with an IR camera that has an analog output, one can choose how to digitize the signal through the choice of a frame grabber, instead of being locked to the Ethernet output. After removing Gobi-640 and A35Sc from the list, it is readily seen that the Tau2 640 and Tau2 336 require less power, and weigh less than the Eye-R25. This means that if pricing is not a criterion, the Tau2 640 is the infrared camera that is best suited for use in the payload. Now, since pricing actually is a factor, the Tau2 640 was acquired for the use in larger UAVs that are robust and not likely to crash. The cheaper version (Tau2 336) was acquired for the use in smaller UAVs such as miniature helicopters. The Tau2 IR cameras comes with an interfacing module enabling power over USB and a BNC video output (analog video).

Now, the final decision that has to be made is what focal length the lens on the camera should have. Because the camera is thought to be mounted on the UAV in a fixed manner, i.e it is not allowed to turn/rotate about its own axis, the *field of view* of the camera is crucial for the success of detection. However, there is a conflict between having a large field of view and actually being able to detect

[3] A frame grabber is a device that converts analog video to digital video, usually connected via either an USB port or an Ethernet connection.

Table 4.4: A table listing focal length vs. field of view.

Camera \Focal Length	**7.5mm**	**9mm**	**13mm**	**19mm**
Tau2 640	$90° \times 69°$	$69° \times 56°$	$45° \times 37°$	$32° \times 26°$
Tau2 336	$45° \times 35°$	$35° \times 27°$	$25° \times 19°$	$17° \times 13°$

objects on the ground. This is because a small focal length will yield a large field of view, but everything in the image will appear small. On the other hand, a large focal length will yield a small field of view, but objects in the image will appear large. The focal length of the lens and the corresponding field of view for the Tau2 640 and Tau2 336 are listed in table 4.4. To balance out the perceived object size and the field of view, a field of view of at least $30° - 35°$ was considered necessary. This will allow the UAV to roll up to $\sim 25°$ while most of the objects still will be present in the image. In other words, a focal length of $13mm$ and $9mm$ was chosen for the Tau2 640 and Tau2 336, respectively. Now, consider the case where the infrared camera is mounted on the side of the UAV, pointing 45° downwards, and the UAV is operating at an altitude of 50m. In this scenario, a normal sized human (in the center of the image) will be projected onto the image plane as 11 pixels wide and 20 pixel high for the Tau2 640, and 8 pixels wide and 16 pixels high in the Tau2 336. This is a bare minimum to achieve good detection and tracking. The mathematics behind this example is given in appendix B.

4.1.3 IP Radio / Communication

Reliable inter-communication in the payload and communication between the payload and the ground station is equally important. However, there is one major difference between the two. That is, within the payload, a *lossless communication layer*[4] can be assumed present. This is not the case for the communication layer between the payload and the ground station. That is, messages sent over large distances, to and from fast moving targets, will be subject to random failures. E.g the Doppler effect, weak signal or in some cases no signal (due to physical blocking) are all problems that has to be explicitly dealt with. Hence, a redundant communication system must be used to make sure that all packages reaches their destination. In this section, an implementation of an inter-communication layer in the payload is suggested, followed by a review of an existing transfer protocol,

[4] A lossless communication layer is a communication layer where no information is lost. Hence, no information has to be sent more than once in order to arrive at the receiving end.

namely *intermodule communication* (IMC), for application in IP radio communication, i.e communication between the payload and the ground station. Note that the *hardware* needed for long distance communication (exceeding the Wi-Fi range) was not evaluated, as it was considered already present in the UAV when needed.

Now, to achieve reliable, fast and efficient communication between the components within the payload, the easiest solution is simply to connect every component to the single board computer. However, most SBCs have a very limited amount of input/output possibilities, hence this easily becomes an inefficient approach. Connecting every component to the single board computer will also increase the risk of a full system failure because if the single board computer fails, all contact is also lost with the other components. A better solution, in many cases, is therefore to connect every component to a common communication layer. This will make the data flow of each component independent of each other, and a single failure will not affect the system in the same way. A robust and effective communication layer for components that are physically close to each other, can be realized through the use of an *Ethernet switch*. Ethernet switches use the TCP/IP protocol in order to manage the data flow of the connected components, and are interfaced through RJ45 cables. Such switches may also be designed to be both light weight and require very little electrical power. These are attributes that make them ideal for the use in UAVs. Furthermore, the Ethernet interface is very common and next to mandatory on single board computers, which in turn means it is easy to find alternative components that fit into the communication scheme of the payload. In the search for an Ethernet switch suitable for use in the payload, the following characteristics were considered most important

- Sufficient amount of Ethernet ports (at least 4)
- Small in size (less than $10cm \times 10cm \times 5cm$)
- Low weight (maximum 150g)
- Low power requirement (max 5W)
- Preferably a 5VDC requirement

Since almost all switches with these attributes work in almost exactly the same way, the reader is not bothered with a list of different possible choices. Instead, the specifications of the switch that was found to best satisfy the above mentioned requirements, the *TRENDnet 5-Port Fast Ethernet Switch*, can be seen in appendix

H. Note that it is only the bare minimum circuit board that should be used in the payload (i.e no chassis).

Having established a way to make the components in the payload communicate in an efficient manner, one question still remains. How should the payload communicate with the ground station? The answer to this question is the *intermodule communication* protocol. The IMC protocol achieves inter module communication by the exchange of messages that can be recognized and interpreted by all the participating modules. A module, in this case, refers to any module that can partake in wireless radio communication, e.g a ground station, an UAV or simply just a radio transponder. Radio transponders are devices that can be used to transmit the same signal as it receives, effectively extending the maximum distance that the IMC data packets can be sent. Now, let us consider the case where a ground station wants to send a command to the autopilot in an UAV. The first thing that is done, is that the ground station creates the command internally, i.e it creates a command that the autopilot can understand. This may be a message on the form "roll +5 degrees". Then, a header of meta data is added to this message. The form and size of this header is specified by the IMC protocol, and includes information such as

- The source of the message (the ground station)
- The destination of the message (the UAV)
- The type of message (command to the autopilot)
- The length of the message
- A time stamp

Some header data might be added or removed depending on the type of message being sent. The process of adding a header to the message is referred to as *serializing* the message. This is done by an IMC manager located at the ground station, and is the last step before the message is sent over the network. The network might be a combination of Ethernet networks and radio networks, but the communication is restricted to the case where both the source module and the destination module operate through an Ethernet interface. This is achieved by sending *IP radio messages* over the radio networks, which in short is the concept of converting TCP/IP packets to radio waves, and radio waves to TCP/IP packets. Exactly how this is done is out of the scope of this thesis, but it is a well known

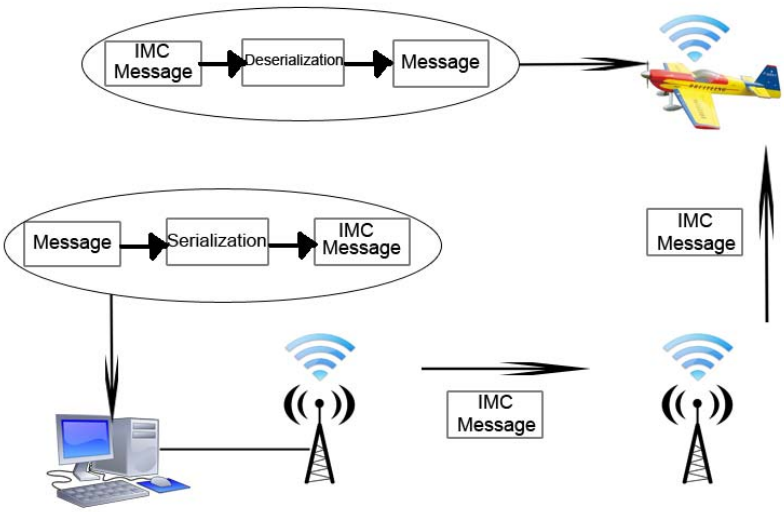

Figure 4.1: The ground station and the UAV communicates through the IMC protocol. This is a protocol that takes any kind of message and *serialize* it, effectively adding a header enabling the IMC protocol to keep track of the messages and also check if the messages arrive at their destination. At the destination, the IMC message is *deserialized*, which is a process where the original message is reconstructed. It should be noted that every node in the network (in this case there are 4 nodes), are responsible for propagating the message through the network, making sure it arrives at its destination.

method to send data over large distances. When the IMC message is sent over the network, it is usually converted to an IP radio message close to the ground station, and propagated through the network using every module in the network available. This is referred to as a *mesh network*[5]. When the IP radio message finally reaches the UAV, it is converted back to a TCP/IP packet and sent to the IMC manager located in the UAV. The IMC manager *deserializes* the message, obtaining the original command message sent from the ground station. The original message is then sent the correct internal module (in this case the autopilot), before the IMC manager sends a data packet back to the ground station with an acknowledgment that it has received the message. The process of sending a message through the network using the IMC protocol is illustrated in figure 4.1. It should be noted that the IMC protocol also has instructions for how to handle cases where information or data packets are lost. For a closer look on how the IMC protocol operates and how it achieves fault tolerant network communication, the reader is referred to [1]. In the present thesis, the case where the distance between the ground station and the UAV exceeds the range of wireless networks is not considered, hence no radio transmitter/receiver hardware was needed.

4.1.4 Power Supply

In the search for a suitable battery pack for the payload, some characteristics of different types of batteries were gathered. Specifically, a list of the energy density (Wh/kg) and specific energy (Wh/L) for different types of batteries was made. The result from this work can be seen visualized as a graph plot in figure 4.2. From this figure it is readily seen that it is the *lithium* and *zinc-air* battery types that offer the most electrical energy with respect to both size and weight. Further research revealed that the recommended discharge rate for each $1.4V$ zinc-air battery cell is in the range of $1-40mA$, which would require too many battery cells to successfully drive the payload. However, *lithium batteries* are known to be able to deliver high discharge rates, hence they were considered as a good choice for supplying the payload with electrical power. Furthermore, considering the power requirement of the already mentioned components, the lithium battery pack had to feature the following

- Low weight ($\leq 200g$)

[5] A mesh network is a network where every participating module must not only capture and disseminate its own data packets, but also serve as a relay for other modules. I.e, the participating modules must collaborate to propagate the data packets in the network.

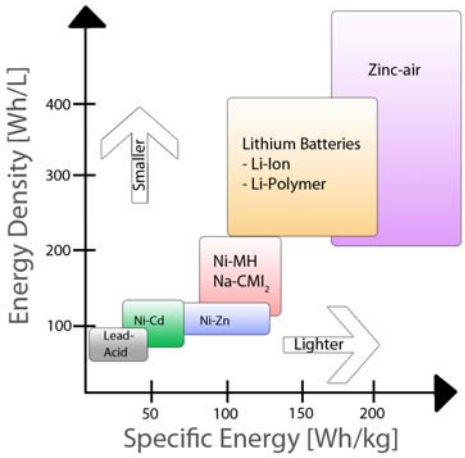

Figure 4.2: Energy density against specific energy for some common battery types (data from [42]).

- Small size (less than $10cm \times 10cm \times 5cm$)
- Decent capacity (at least 2000mAh)
- Regulated $5V$ output
- Able to provide enough Watt ($\geq 15W$)

The requirement of having a regulated $5V$ output is there because both the PandaBoard *and* the Tau640/335 IR camera require a steady voltage input of $5V$. In the case where an Ethernet switch is required, the switch mentioned in section 4.1.3 also require an input voltage of a steady $5V$. The PandaBoard is especially sensitive to this, and typically shuts down if the voltage is lower than $4.8V$ or higher than $5.2V$. Now, most lithium ion (Li-Ion) and lithium polymer (Li-Po) battery cells has a nominal voltage of $\sim 3.7V$, but this voltage typically decreases as the battery cell discharges. Hence, to have a steady voltage output, a voltage regulator is needed.

In recent years, a type of battery pack often referred to as *external power packs* has become increasingly popular. External power packs are battery packs initially made to charge smart phones and tablets, but they inherit several characteristics

Table 4.5: Summary of the Search for Different Power Packs

Power Pack	**Capacity**	**Total Max. Current**	**Output**	**Weight**	**Price**
Biltema PowerPack	2000mAh	1A	5V USB	75g	$27
IT-CEO	3000mAh	1A	5V USB	90g	$25
PB5K	5000mAh	2A	2x5V USB	140g	$45
Duracell PPS2	1150mAh	0.5A	5V USB	41g	$20
Sinotek OEM	10000mAh	3A	2x5V USB	195g	$20

that are suited for the use in the payload designed in the present thesis. For example, most power packs come with either a $5V$ or $12V$ (some power packs feature *both*) regulated output. Furthermore, most power packs come with built in protection against short circuits and over-drain[6], making them more safe in use. These are must-have features, as e.g Li-Po batteries are known to be very unstable and easily set fire if misused. Because of this, the use of power packs to supply the payload with power was considered a good choice. The wide variety of available power packs also makes it easy to find a combination of power packs that is able to drive the payload regardless of how many components and modules it contains.

The search for different suitable power packs is summarized in table 4.5. From this table, the Duracell PPS2 was excluded because of its low maximum current draw. $0.5A$ per power pack means that at least 2 such packs are needed only to power the PandaBoard. Also, considering the price and capacity compared to the alternatives, this is not an optimal solution. The Sinotek OEM was excluded because it has a relatively high weight, which in turn makes it impossible to use in some UAVs. The IT-CEO is very similar to the Biltema PowerPack, with the difference being that the IT-CEO is 15g heavier. This makes the IT-CEO excessive, and was therefore also removed from the list. Hence, the list was shortened down to only two power packs; the PB5K and the Biltema PowerPack. Notice that using two Biltema PowerPacks is very similar to that of using one PB5K. The difference between these two solutions is that the PB5K is 10g lighter, which is easily outweighed by the fact that using the Biltema PowerPacks, a *third* PowerPack can be added to the payload without exceeding the limit of 200g by more than 25g. Furthermore, removing the chasing of the PowerPack, it was revealed that the voltage regulator and the battery only weighs $45g$. This means that it is possible to use *four* such packs and still not break the limit of 200g. Hence, the Biltema PowerPack was considered the best choice for the payload. The specifications for

[6] *Over-drain* occurs when a battery is discharged at a higher rate than what is considered safe and/or correct.

this component is listed in appendix G.

4.1.5 Video Camera

In the payload designed in the present thesis, the video camera module plays only a passive part. That is, the video stream from the video camera is not actively used in the object detection and tracking algorithms, which in turn means that the most important thing at this point is not exactly *which camera*, but rather *how* the module should be integrated in the payload. Hence, the exact choice of camera was not a prioritized activity. However, in the search for a video camera, these were the requirements considered important

- Low weight (maximum $100g$)
- Small in size (less than $10cm \times 10cm \times 10cm$)
- Low power requirement (max $5W$)
- At least the same field of view as the Tau2 IR camera
- At least the same resolution as the Tau2 IR camera

In this regard, the GoPro Hero2 video camera (the specifications are listed in appendix E) is a solid choice. Not only does it satisfy all the above mentioned requirements, but it has an internal battery, making it very easy to use. This adds to the simplicity of the payload, making it applicable in virtually any payload. It also has an analog output, making it easy to connect the camera to the SBC in the same way as the Tau2 IR camera, i.e with a frame grabber. It should be noted that there exist many cameras more suited for the payload than the GoPro Hero2, but it is good enough as a temporary solution.

4.1.6 GPS/INS

Since non of the object detection and tracking algorithms that are implemented in the present thesis relies on a navigation module, the procurement of such a module was not considered very important. However, a review of possible solutions for such

a module still is important. This is especially so because future implementations of the object detection and tracking algorithms probably will need some navigational data at some point. Hence, in this section, a type of devices suited for use in the navigation module in the payload is suggested.

In recent research on navigation in UAVs, one of the most widely used navigation technologies is the combination of a *Global Positioning System* (GPS) and an *Inertial Navigation System* (INS) device [15]. The INS device operates without any inputs or external signals, and provides a complete set of navigational parameters. That is, it provides the position, velocity and attitude of the UAV with a high sampling frequency (i.e many updates per second). Now, the problem with using only an INS is that systematic errors will grow over time. For example, if the measured velocity is constantly off by +1m/s from the real velocity, the positional error will only increase over time. This implies that it is a good idea to use GPS measurements to correct the outputs from the INS. This is usually achieved by the use of an *Extended Kalman Filter* (EKF), which is a filter that is used to estimate non-linear processes.

Due to its wide spread use in navigation for UAVs, it was decided that the navigation module in the payload should be realized through the use of a GPS/INS device. There exist many GPS/INS devices suitable for the use in UAVs, most of which have an USB interface. This is very convenient since most SBCs have at least two USB ports. It should also be noted that some GPS/INS devices have implemented the extended Kalman filter internally, removing the need to implement this on the single board computer. This is a very nice feature, as it leaves more computational power free for the object detection and tracking algorithms. However, these devices are usually considerably more expensive than the devices that do not filter the signal internally. Finally, it should be noted that the INS device should be physically placed as close to the center of gravity of the UAV as possible, because most standardized INS devices model the UAV as a point mass. Hence, it assumes that all of the motion dynamics happen at one central location. This would not be true if the INS is placed far from the center of gravity, as e.g the UAV rolling would induce an extra rotational velocity in the INS.

4.2 Software

The different types of software that is needed for the payload, can be divided into three different categories. The first category is the operation system (OS) of the payload, i.e the software that handles the input and output of the system, and is responsible for the intercommunication between different tasks in the payload. The second category is that of object detection and tracking. There exist several software solutions that supply the programmer with a framework for working with computer vision, and choosing the appropriate software solution can simplify the implementation of the object detection and tracking algorithms greatly. The third and final category is that of navigation and communication. This means the software responsible for controlling the UAV and communicating important information with the ground station.

Now, the choice of the appropriate software for the payload and ground station is crucial for the performance of the over all system, hence the task was not taken lightly. Furthermore, it was important that the software was *reliable*, *easily modified* and *up to date*. It should be reliable because the payload is required to be operational at all times during flight. It should be easily modified because it should be easy to add new features or modify existing ones, and it should be up to date because computer science evolves at an impressive pace. Several different software solutions were evaluated, and following is the software from each of the three categories that were considered to be best suited for application in the payload.

4.2.1 Ubuntu

Ubuntu is an open source operating system based on the Linux kernel. It is one of the most popular Linux distributions not only for personal use, but also in servers. The reason for Ubuntu being a good choice for the use in the payload, is that it is very easily customized to only feature the bare amount of functions that is needed for the payload to function properly. This will, in turn, make the computer vision algorithms run faster, as more computational power is available to these algorithms. It is also a very reliable OS, as it has become quite stable after 8 years of development. Furthermore, since the OS is so popular, it is continuously updated. Hence it also has support for many external USB components, such as frame grabbers and different GPS devices. Ubuntu is also a widely used OS for ARM systems (such as the PandaBoard) in general, resulting in a variety of Ubuntu

distributions that are optimized with respect to the ARM architecture. All of these factors makes Ubuntu an ideal operation system for the PandaBoard.

4.2.2 OpenCV

The Intel Open Computer Vision Library, referred to as OpenCV, is a collection of many computer vision related algorithms. It also integrates some existing video capturing libraries, effectively simplifying the process of reading images from a video camera. The library features convenient data structures for storing images, and supplies the programmer with a framework for working with HOG-descriptors, Haar-like features and classifiers. Furthermore, compared to other similar libraries, OpenCV is known for its speed and focus on real time computing, while at the same time maintaining a large diversity. This makes OpenCV ideal for creating and experimenting with new computer vision algorithms. The library is also under constant development, with updated versions being released on a regular basis.

4.2.3 Dune

DUNE Unified Navigational Environment is a runtime environment for vehicle onboard software developed by the *Underwater Systems and Technology Laboratory* (LSTS) located in Porto. It provides an operating system and architecture independent platform for control and navigation of different types of unmanned vehicles (including UAVs). This means that if the OS or architecture of the payload changes (e.g a change of single board computer), Dune can remain unchanged, and will work out of the box in the new payload. This is a huge advantage, as the process of making hardware components communicate with each other can be a tiresome process. Dune includes drivers for sensors and actuator access, communication with a ground station and has interfaces to many popular autopilot solutions. In addition to this, Dune has inbuilt functions such as the extended Kalman filter and similar algorithms. When Dune is communicating with a ground station, it uses the IMC protocol described in section 4.1.3. It should be noted that Dune is based on running separate *tasks*, where each task has its own unique designated responsibility. For example, there is one task for reading the GPS/INS output, and there is another task for sending commands to the autopilot. Adding and removing tasks is trivial, hence Dune is highly modularized. This makes it easy to adapt Dune to function optimally in any payload, in any unmanned vehicle and any situation.

4.2.4 Neptus

Neptus is a ground station software developed by the creators of Dune. It is a software that enables the operator to command and control fleets of different types of unmanned vehicles (including UAVs). Neptus can also be used to create and manage mission plans, and is an excellent software to use when executing these plans. Real-time information about the unmanned vehicles, such as CPU load, battery capacity and free hard drive space is also made available to the operator. In addition to this, other sensor information such as GPS/INS data and video streams are easily interfaced to Neptus, effectively gathering and displaying all available information in one and the same place. As with Dune, Neptus achieves remote communication through the use of the IMC protocol, and command and control of the vehicles are obtained by exchanging messages with Dune. Using the Neptus-IMC-Dune software tool chain has proven a great success, and has been thoroughly tested by LSTS. This makes this combination of software a solid solution for the payload designed in the present thesis.

4.3 Setup Configuration 1

The first setup configuration is designed with simplicity in mind. This is because it is a configuration that even small and light UAVs should be able to handle. In other words, this means that one of the highest priorities of this setup is *low weight*. It should be noted that the maximum payload weight for different small UAVs may vary, hence the design is made such that the functionality in the payload is restricted by the weight requirement. To be more precise, the components featured in the payload are added one by one until the maximum payload weight for the UAV is reached. The following is a list of the components in a prioritized order, i.e the order in which the components should be added to the payload.

- Battery (2xPowerPack)
- Single Board Computer (PandaBoard)
- Infrared Camera (Tau2 336)
- Frame Grabber (EasyCap DC60)
- Video Camera (GoPro Hero2)

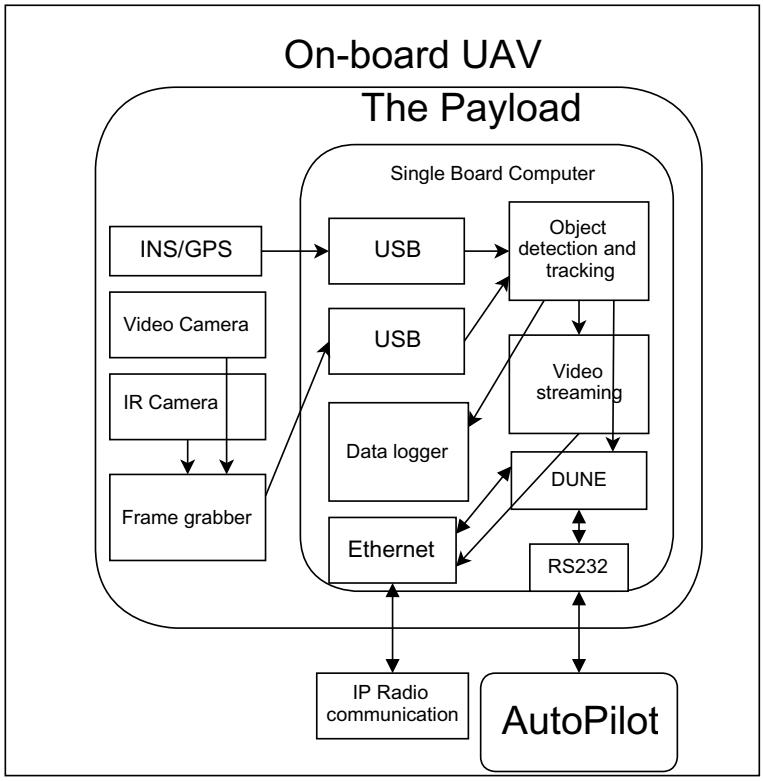

Figure 4.3: Connection diagram for setup configuration 1. Notice that Dune is running on the single board computer in the payload.

- GPS/INS

This adds up to a total weight of $\sim 400g$, if the weight of a GPS/INS device is excluded. Furthermore, it is assumed that some means of communicating with the payload is already present and available for use. In some scenarios and applications, this will require a sophisticated data link between the payload and the ground station, but technologies and solutions for this was not evaluated and out of the scope of the present thesis.

Now, the connection diagram for these components is illustrated in figure 4.3, where

the IP Radio / communication link can simply be a wireless network connection. The object detection and tracking algorithm, data logger and video streaming modules are processes running in the CPU on the PandaBoard, communicating through reading and writing to files. The data logger is responsible for logging the position of the tracked objects to the hard drive. The video streaming module is responsible for streaming the video from the infrared camera to the ground station through the use of the IMC communication protocol. Notice that, in this configuration, *Dune* is also running locally on the PandaBoard. Dune is responsible for communicating with the autopilot (if any) through the RS232 port located on the PandaBoard. Because Dune is so easily modified, it can also include a part that communicates with the object detection and tracking algorithm. This makes it possible for the object detection and tracking algorithm to control the UAV by sending commands to the autopilot through the communication channel with Dune. As previously mentioned, Dune is also responsible for communicating with the ground station, e.g receiving commands or reporting the position, velocity and status of the UAV.

The power diagram of the payload is illustrated in figure 4.4. Note that the infrared camera has a VPC module connected, separating the Tau2 connection into one power part (through USB) and one analog signal part (BNC). In addition to this, the PandaBoard can be powered through its USB2.0 OTG connection, hence, all of the devices are either self powered (Hero2 video camera) or powered through USB. This makes it very easy to power the payload in an efficient way, without the need for solutions tailor made for this specific setup. This enables the payload to be highly modularized, and adding or removing components can be done without too much redesign.

4.4 Setup Configuration 2

The second setup configuration is a little more feature rich, and also more robust to single module failures than the previous setup configuration. This is achieved by adding an Ethernet communication layer between most of the components, and also outsourcing the navigation module of the previous setup (Dune and communication with the autopilot) to a second single board computer. In this configuration it is the navigation module that communicates with the ground station, now through an IP Radio communication device as described in section 4.1.3. The navigation module is also communicating with the payload on-board the UAV through the Ethernet communication layer now present in the payload. By designing the total payload in

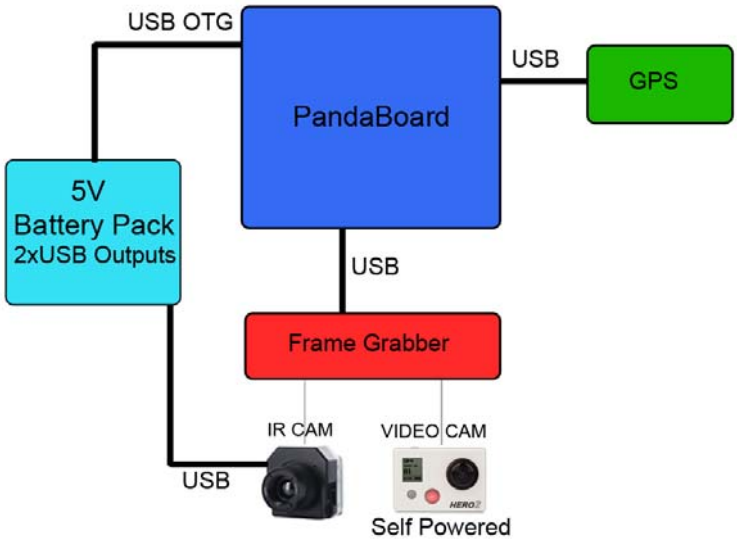

Figure 4.4: Diagram showing how the battery pack is connected to each component for the first setup configuration. The two USB outputs from the battery pack are wired in parallel. Note that the frame grabber and the GPS use the USB channel as both its power source *and* the data communication channel

this way, a single component failure does not have to affect the total functionality of the payload very much. E.g if the PandaBoard were to malfunction during flight in setup configuration 1, all communication with the ground station would be lost. Furthermore, the communication with the autopilot would be lost, in effect making the flight operation very unsafe as there would be no way to resume control over the UAV. If the PandaBoard were to malfunction during flight with setup configuration 2, the navigation module would still be intact, able to control the UAV in a safe way. The connection diagram of this setup configuration is illustrated in figure 4.5.

The setup configuration described above can be realized using the following components

- Battery (4xPowerPack)
- Single Board Computer (PandaBoard)
- Ethernet Switch (TRENDnet Switch)
- Infrared Camera (Tau2 640)
- Frame Grabber (2xAxis M7001)
- Video Camera (GoPro Hero2)
- GPS/INS

This setup weighs a total of $\sim 750g$, if the weight of a GPS/INS device is excluded. Now, in particular, this setup configuration is an enhancement of the setup configuration described in section 4.3 in that it relieves the PandaBoard of some work. This is mainly because Dune is no longer running in the CPU of the PandaBoard, but also because the Axis M7001 frame grabber does the encoding internally, while the previous frame grabber (EasyCap), leaves that job for the PandaBoard.

To power the payload in this setup configuration, a little more work than the USB-solution from the previous section has to be done. To be more specific, the power interface of the Axis M7001 encoder is Power over Ethernet[7] (POE), making it necessary to inject the Ethernet cable connected to this encoder with electrical

[7]In 10/100 Mbps networks, there are one unused pair of conductors in the Ethernet cables. When this pair is used to provide electrical power to a device, it is said that the device is *Powered over Ethernet*

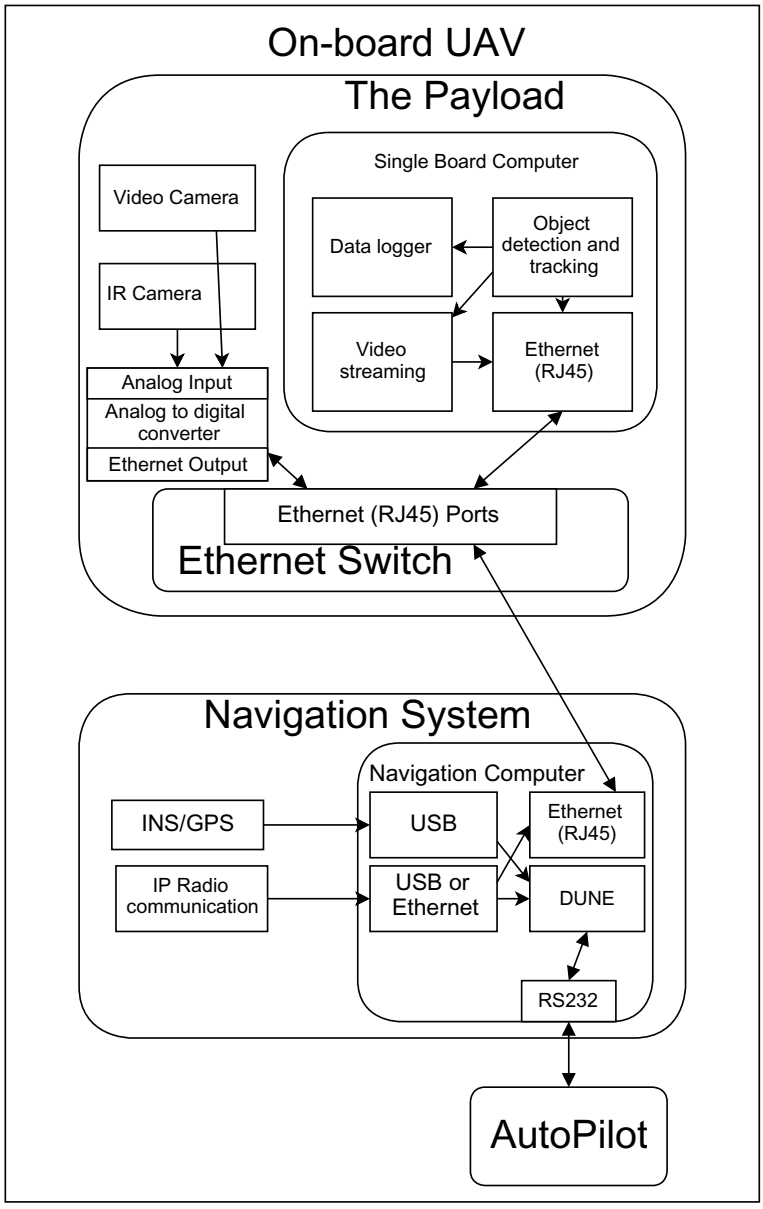

Figure 4.5: Connection diagram for setup configuration 2. Notice that Dune now is running on a separate computer.

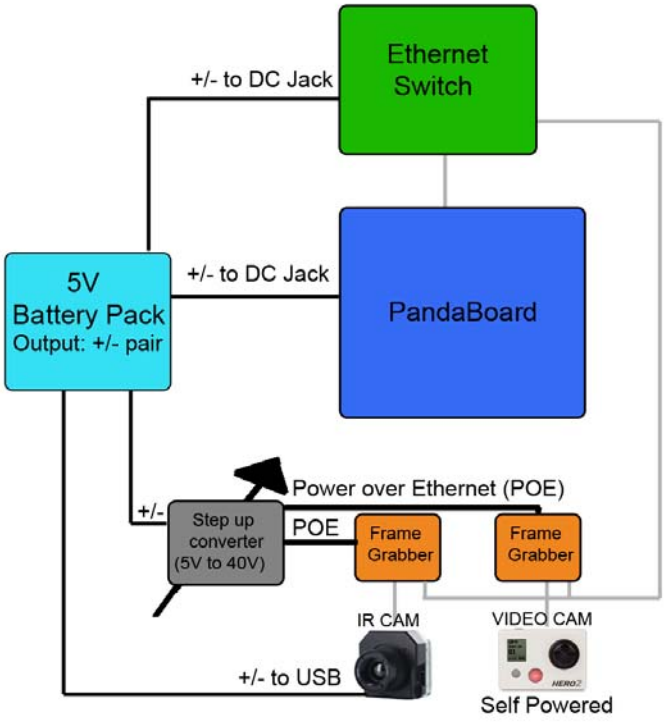

Figure 4.6: Diagram showing how the battery pack is connected to each component for the second setup configuration. Every component is wired in parallel to supply each component with 5V. The data connection between the components are illustrated with gray lines.

power at some point. Furthermore, the voltage needed to comply with the requirement of this device is in the range of 36V-48V. This in turn, means that a step-up converter has to be present, converting an input voltage of 5V up to at least 36V. It should be noted that the internal encoder board of the Axis M7001 requires only 1.2V, 1.8V and 3.3V (connected at different pins). However, the design of a circuit board able to supply the encoder with these voltage inputs was out of the scope of this thesis, hence the POE interface was kept. The Ethernet switch must also be powered, requiring 5V power through a DC jack. The total power diagram of this setup can be seen in figure 4.6. The power scheme for the navigation module is not presented, as this module is assumed to already be integrated and ready for use in the UAV.

Note that the data logger and video streaming modules are exactly the same as earlier, but that the GPS now is connected to the navigation module. This means that, in order to make the data available to the object detection and tracking algorithm, the GPS data has to be broadcasted over the local Ethernet communication layer. Similarly, the video streaming module does not have direct access to the IP Radio communication channel in this setup configuration, hence it has to send the video stream over the local network. Dune should then include a task that reads this stream, and forwards it to the ground station through the communication channel. Configuring Dune to do these tasks is trivial.

This page intentionally left blank.

Chapter 5

Experiments

Having designed the payload, the next step is to implement and test it together with an object detection and tracking algorithm. Furthermore, to consider the design and implementation successful, it was desirable to arrive at a solution that can be used in the field for *real-time multiple human detection and tracking*. The implementation should also be able to stream video, log the position trajectories of humans and control the UAV. The last part is only required to be simulated, i.e the payload is not required to actually communicate with the autopilot. However, the payload should be able to decide in which direction the UAV should move to keep track of the humans located in the field of view of the camera. Now, the process of creating such a system was divided into the following steps.

1. Train classifiers based on either the SVM/HOG or the Boosted Cascade/Haar-like features combination.
2. Implement and test the object tracking algorithm described in chapter 3.
3. Implement and test the following functions
    - Video streaming to the ground station from the payload.
    - A data logger.
    - Algorithm that simulates control over the UAV.
4. Verify that setup configuration 1 is working as intended.

5. Optimize the performance of the object tracking algorithm when it is running on the SBC.

6. Field test of setup configuration 1 with the implemented object tracking algorithm.

The following chapter is a description of the methods used and the choices made to complete all of the above mentioned steps.

5.1 Classifier Training

Before object detection and tracking can be performed, some classifiers have to be trained. Following the theory described in chapter 2, two fundamentally different classifiers can be trained and evaluated. These two classifiers are a support vector machine classifier based on the histogram of oriented gradients feature representation (SVM/HOG classifier), and a boosted cascade classifier based on Haar-like features (BC/HL classifier). To evaluate the performance of these two classifiers from a common ground, a training set was created. This was done by acquiring 3 long-wave infrared video sequences of pedestrians walking on a street. The video sequences were acquired at different days, but under the same weather conditions and temperature. This is because in thermal imaging, humans appear differently depending on their temperature relative to the environment. In a cold winter night, humans will appear in the image as white (hot compared to the environment), but black (cold compared to the environment) on a warm summer day. Hence, in reality this is equivalent to detecting two different objects. Now, the training set was created by extracting positive and negative examples from each frame in the recorded video sequences. The *positive training examples* consisted of 500 images of pedestrians at an upright pose and the *negative training examples* were 3000 images randomly sampled from the background of the same video sequences. Furthermore, a *test set* of 150 images was created from a fourth video sequence. This was done because the classifier performance should be evaluated on novel data, and *not* the data that were used to train the classifiers. A small sample of positive and negative examples from the training set can be seen in figure 5.1

When a classifier is used to identify objects in an image, a search window is used. This search window is of a fixed size, and it gradually moves from one end of the image to the other, classifying each search window as either an object or not an

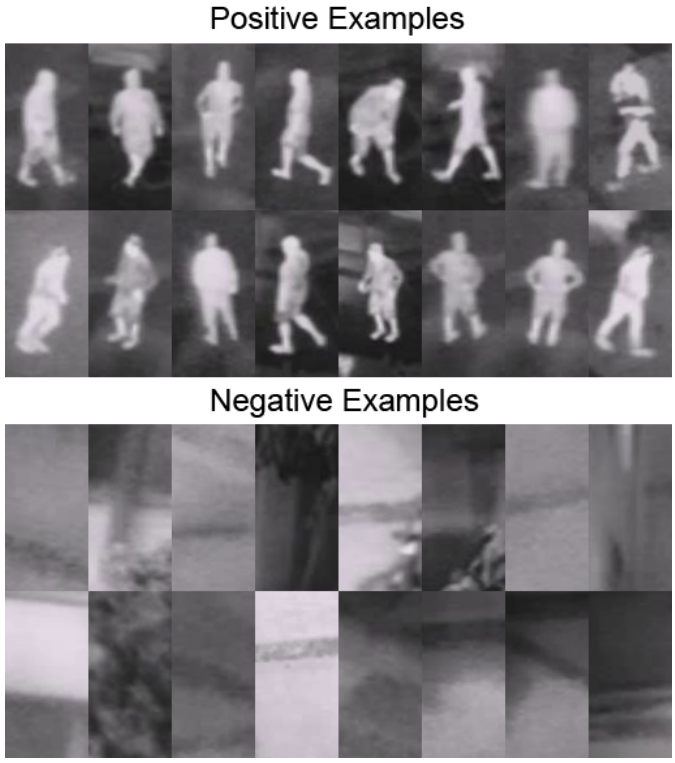

Figure 5.1: A small sample of both positive and negative examples from the training set used to train the classifiers.

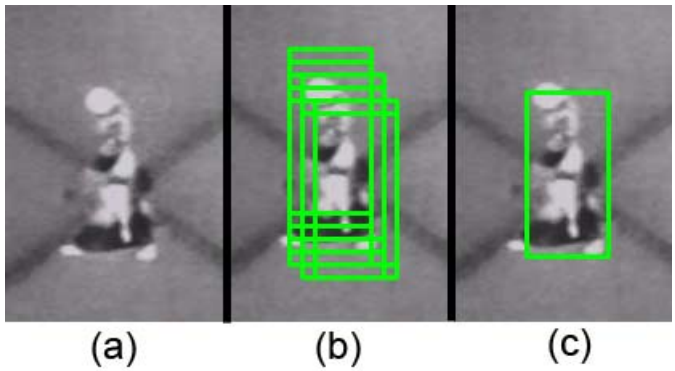

Figure 5.2: (a) shows the original image, and (b) illustrates how several search windows in the vicinity of a pedestrian usually are classified as positive (represented by a green rectangle). (c) shows a filtered version where positive classifications close to each other can be merged into only *one* classification.

object. Now, since the search window moves gradually, several search windows in the vicinity of an object may contain the object. As illustrated in figure 5.2, this may result in that the classifier returns several positives for the same object. Hence, it is often a good idea to demand that the classifier classifies n number of search windows, in a certain vicinity of each other, as positives before concluding that an object is present. It should be clear that this will reduce the number of false positives, as the classification of a false positive is not necessarily correlated with a certain area of the image. However, it should also be noted that increasing the number of positives n required for the final classification to be positive, will *reduce* the percentage of objects found. In other words, by varying the number n and performing classification on the test set, the classifier will have a varying true positive rate (percentage of objects found) and false negatives rate (percentage of negative examples classified as positives). By plotting the true positive rate (TPR) against the false positive rate (FPR) for different values of n, a *receiver operating characteristic* curve (ROC curve) is plotted, which is a very efficient way to compare the performance of classifiers against each other.

The training and testing process for each of the two classifiers used in the present thesis are described in the following section.

5.1.1 Training and Testing the SVM/HOG Classifier

The first thing that has to be decided when a SVM/HOG classifier is going to be trained, is the size of the HOG descriptors. This is decided by the following variables

- The resolution of the training examples
- The number of pixels within each cell
- The number of cells within each block
- The number of orientation bins
- The number of cells that neighboring blocks have in common

These variables were set to the same configuration as that which was found to work best in [14], i.e a resolution of 64x128 pixels, 8x8 pixels in each cell, 4 cells in each block, 9 orientation bins (evenly spaced over $0° - 180°$) and an overlap between each block of 1x1 cells. This means that the HOG descriptors will have $\frac{64-16}{8} + (\frac{16}{8} - 1) = 7$ blocks in the width, and $\frac{128-16}{8} + (\frac{16}{8} - 1) = 15$ blocks in the height. This means in turn that the whole descriptor will be $7 \times 15 \times 4$ cells, and with 9 orientation bins this means that the full descriptor will be a vector of length $7 \times 15 \times 4 \times 9 = 3780$. This is a suitable number of dimensions for the use in conjunction with a support vector machine for creating a classifier.

To perform the training of the classifier, the above mentioned HOG descriptor was calculated and labeled for each example in the training set. The descriptors were either labeled +1 (a positive example) or −1 (a negative example). Then, the optimization problem described in equation 2.12 was set up and solved using *SVMLight* [22]. This resulted in the creation of a separating hyperplane, described as a vector of the same length as the HOG descriptors, that is 3780. To evaluate the effect of increasing the size of the training set, two other classifiers were also trained. One of these classifiers were trained by including the mirrored version of the original training set in the training process. The second was also trained in this manner, with an additional 500 negative examples.

To test the performance of the classifiers on the test data, each test image (640x512 pixels) was searched for pedestrians. This was done by using a search window of

the same size as the resolution used for training (64x128), starting at the top left corner $(0, 0)$ and sliding it over the test image, moving it 16 pixels at a time. At each position, the HOG descriptor for the search window was calculated. In practice, this was done by first calculating the integral histogram image (equation 2.7) for all of the 9 different orientation bins (this step is only done *once* per test image). Then, the HOG descriptor for a given search window was constructed by calculating the total histogram of oriented gradients for all the cells that make up the total descriptor (i.e all of the $7 \times 15 \times 4$ cells). The total histogram for a cell was found by summing the contribution of the integral histogram to the cell over all the orientation bins. Now, after the HOG descriptor for the search window was obtained, the L_2 distance from this descriptor to the hyperplane created in the training step was calculated. If this distance was negative, the search window was considered *not* to contain a pedestrian. A positive distance however, was assumed to imply the presence of a pedestrian in the search window. Furthermore, the requirement that several search windows in the vicinity of each other have to be classified as positive windows for a pedestrian to actually be present, was introduced. This was done by grouping together search windows that were both closer than 0.2 times the width and height of the search windows to each other *and* classified as a pedestrian. After this grouping was done, only the groups consisting of more than n rectangles were kept. The ROC curve was then created by performing this test for different values of n and calculating the corresponding true positive rate and false positive rate.

5.1.2 Training and Testing the BC/HL Classifier

To train a boosted cascade classifier, the following has to be decided

- The resolution of the training examples
- The number of stages
- Minimum hit rate for each stage
- Maximum false alarm rate for each stage
- The set of Haar-like features to be used

where the minimum hit rate for each stage is the minimum percentage of correctly classified positive examples from the training set. The maximum false alarm rate

is the maximum percentage of negative examples being classified as positive examples. When it comes to the set of Haar-like features to be used during training, there are two options. Either, the set of features can be limited to only vertical and horizontal Haar-like features, or it may also be chosen to include *rotated* Haar-like features as illustrated in figure 2.2.

To train the boosted cascade classifier the resolution of the training examples was decided to be 24×48 pixels. The reason for the small resolution is that Haar-like features are easy to scale to different resolutions. Hence, setting a low resolution during training shortens the duration of the training process, with little difference in the performance of the resulting classifier. Training a boosted cascade classifier based on Haar-like features may take days or even weeks to complete even when the resolution of the training data is low, hence training with larger resolutions should be avoided. The training requires a tremendous amount of calculation because the classifier trainer has to try an overwhelming amount of different Haar-like features before the most discriminative feature is found. Furthermore, this process has to be repeated for each stage. Now, the number of stages were set to 20, and the minimum hit rate and maximum false alarm rate to 0.995 and 0.5 respectively. This means that the total boosted cascade classifier will have a minimum total hit rate of $0.995^{20} \sim 0.905$ and a maximum total false alarm rate of $0.5^{20} \sim 9.6 \times 10^{-7}$ when it is used used to classify the training set. Finally, the set of Haar-like features were chosen to include the rotated variants of the Haar-like features. The choices for these variables is a matter of balancing the time it requires to train the classifier, and the accuracy of the classifier.

The training of the classifier was done by resizing the training set to 24×48 pixels, and by use of the application *opencv_traincascade* included in the OpenCV library. A second boosted cascade classifier which included the mirrored version of the training set was also trained, in addition to a third classifier which was trained with both the mirrored version and the extra 500 negative examples. Now, *opencv_traincascade* is a tool that takes the above mentioned parameters together with a labeled training set as input, and outputs a data file containing the Haar-like features and the threshold for each stage, i.e for each stage a strong classifier of the type described in equation 2.13 is given. *opencv_traincascade* achieves this by applying the *AdaBoost* algorithm, which was described in section 2.2.2, at every stage. It should also be noted that the number of Haar-like features at each stage is a result of the chosen minimum hit rate and maximum false alarm rate. With a maximum false alarm rate of 0.5, the strong classifier at the first stage is only required to filter out *half* of the negative examples, which is not very much. Hence, a small amount of Haar-like features can be used. If the maximum false

alarm was set any lower, more features would have to be added in order to make the first stage satisfy the accuracy requirement. Furthermore, if the minimum hit rate is set high (e.g 0.9999), there are not a lot of room for mistakes, hence many Haar-like features are required to satisfy this accuracy requirement. By the same logic, a lower minimum hit rate will yield a lower amount of Haar-like features.

To test these classifiers, a similar procedure as for the SVM/HOG classifiers was applied. That is, after scaling the boosted cascade classifiers to $64x128$, a search window of the same size was created. As before, sliding the search window over the test images, moving it 16 pixels at a time, the test images were searched for pedestrians. This was done by applying the cascade of strong classifiers on the search window, classifying it as a pedestrian if all the stages in the classifier were passed. If the search window was rejected at some stage, the region was said to not contain a pedestrian. To speed up this process, the technique of integral images described in section 2.1.1 were used to calculate the values of the Haar-like features. Finally, the ROC curves for each of the two classifiers were constructed in the same manner as before.

5.2 Implementing the Object Tracking Algorithm

The implemented object tracking algorithm was heavily based on the theory from chapter 3, and was programmed in *C++*. The *OpenCV* library was extensively used in the process of reading images from the infrared camera, processing these images and also in handling the classifiers. OpenCV has support for both boosted cascade classifiers as well as SVM classifiers, which made it easy to load these classifiers into the *C++* framework. Using the classifiers for object detection and recognition was also simplified through the use of support functions in OpenCV, such as the inbuilt *detect* function. This is an algorithm that checks whether a search window passes through all stages in a cascade classifier or not. The object detection was implemented in the same way as in section 5.1.1 for the SVM/HOG classifiers, and as in section 5.1.2 for the boosted cascade/Haar-like features classifiers. Furthermore, a framework for managing (e.g creating/removing/updating) the Kalman filters was also implemented in *C++*. This worked as a support module for the tracking algorithm, making it easier to track multiple objects at the same time.

Now, the flow of the implemented object tracking algorithm can be seen in fig-

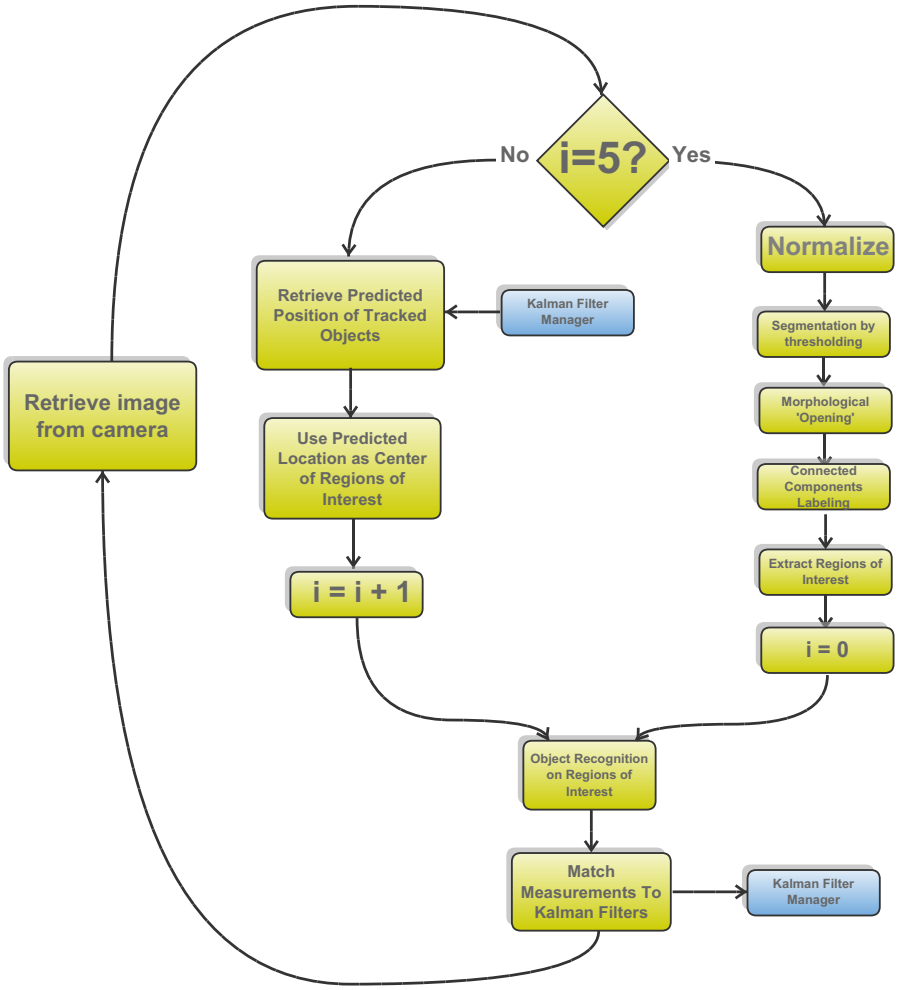

Figure 5.3: A flowchart of the implemented tracking algorithm. The matching of measurements is achieved by solving the global nearest neighbor problem. The Kalman filter manager is responsible for updating, creating and deleting Kalman filters based on measurements from the detection algorithms.

ure 5.3. As illustrated, the algorithm searches the full image for interesting regions once every *5* frames. This is done by applying many of the techniques listed in section 3.2.1, which results in that only the promising regions in the image is forwarded to the detect and match part of the algorithm. Now, in the frames in between each full search, only the regions most likely to contain already detected objects are searched. This is achieved by predicting the position of the tracked objects, and then creating a search window which is centered around this position. The search region was chosen to be 20% bigger than the expected object size, where the expected object size is the same size as the object was last time it was observed. This process is done in order to limit the computational power required to run the tracking algorithm smoothly on the SBC. It should noted that support for multiple sized object detection and tracking can be added to this setup, but that it requires much more computational power than detecting only a single, fixed object size. Multi sized object detection is achieved by having the search window simultaneously slide over different *scaled versions* of the original image. In this way, an object of any size in the original image, will have the same size as the size of the object the detection algorithm is searching for in at least *one* of the scaled versions. Hence, the object detector becomes invariant to the object size. The down side of this is, as mentioned, an increase in required computational power. However, the amount of scaled versions of the original image used will affect both the extra computation needed, *and* the algorithm's ability to detect objects of different size.

The implemented data association (measurement matching) algorithm is a global nearest neighbor solution, created by modifying *hungarian-cpp* [2], an open source solution to the *minimum assignment problem* based on the Kuhn-Munkres algorithm. Furthermore, the Kalman filter manager is the module responsible for either initializing tracking of a new object (adding a Kalman filter in the case of odd measurements), or using the predicted object position as a measurement (no measurement available). The details of how this manager works are illustrated in figure 5.4. In short, the manager checks every fifth processed image frame whether a tracked object has been observed within the last five frames or not. If the object has not been detected within these frames, the tracking of this objects is removed from the manager. It also creates and updates the existing Kalman filters when queried to do so.

To optimize the performance of the object tracking algorithm, it is important to choose appropriate values for the different design parameters. The design parameters of the tracking algorithm are parameters such as the Q, R and P_0 matrices (as described in section 3.1.2) for the Kalman filters. Another parameter is the

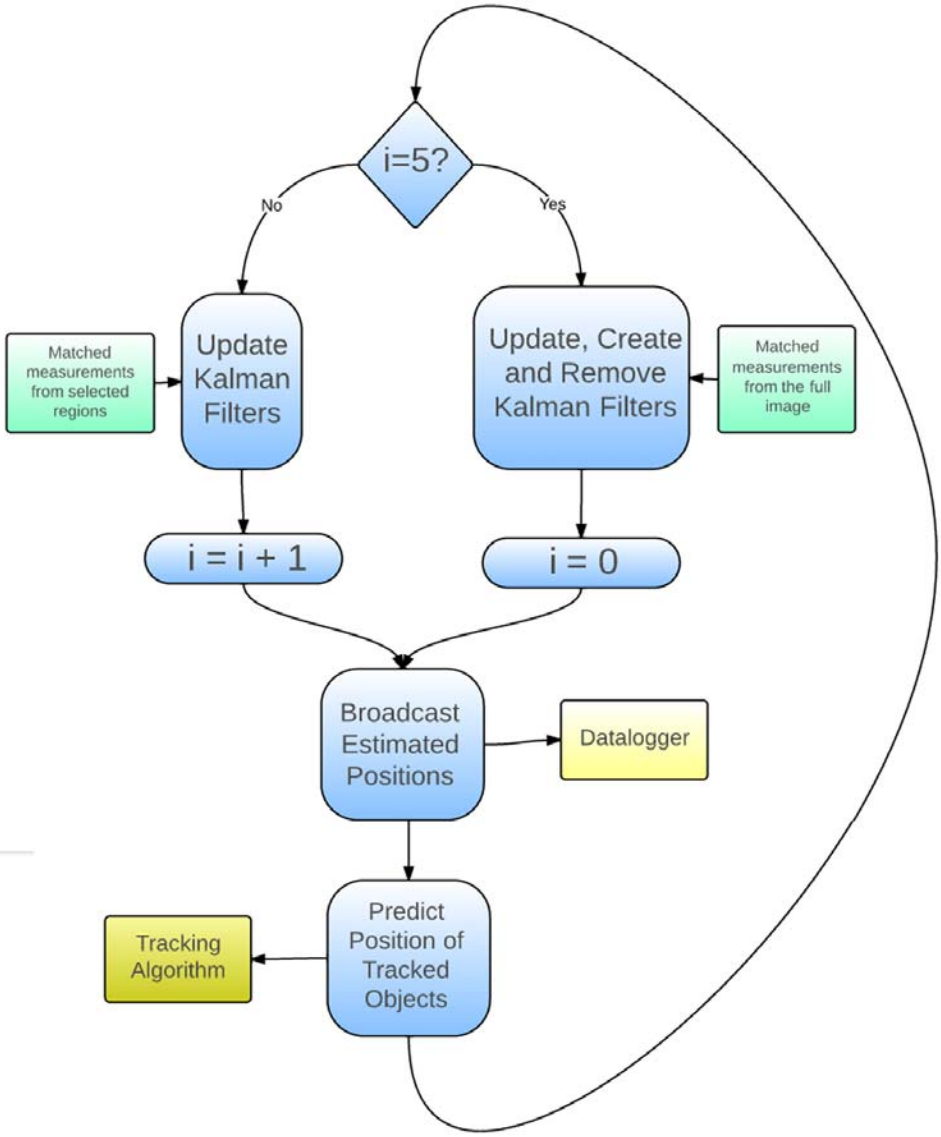

Figure 5.4: Data flow of the Kalman filter manager. The manager supplies the tracking algorithms with predictions, and the tracking algorithm supplies the manager with measurements. This results in an estimate-and-measure tracking approach.

number n of search windows in the vicinity of each other that has to be classified as positive before it is concluded that a human is present. Now, some of these parameters were decided by trying and failing. An example of this is the entries in Q, R and P_0. They were varied performing qualified guesses, and the performance of the tracking algorithm was evaluated for each set of matrices. The performance criteria in this case was the sum of the absolute value of the tracking error (difference between real position and estimated position) in an example video containing a human walking around in a random pattern. The tracking error was found by manually creating the real trajectory path for the human in the video sequence, and then comparing the real position with the estimated position. The choice with the lowest sum of errors was then chosen as the 'optimal' choice. For the variable n, the ROC curves for each classifier was used to decide at which combination of true positive rate and false positive rate the classifier should operate at, when it is applied in the object tracking algorithm. As this approach do not guarantee that the parameters are chosen as a global optimum, it can only be concluded from the tests that the performance of the tracking algorithm is *good enough*.

5.3 Implementing Additional Functions

With the object tracking algorithm functioning properly, real-time tracking in UAV can be achieved. However, without any additional functions it is neither possible to view what the payload is tracking nor review after a test flight how the algorithm performed. This is why some additional features are needed. More specifically, the implementation of streaming from the payload to the ground station, data logging of the positional trajectories of the tracked objects and finally a simple navigation algorithm was of interest.

The video streaming from the payload was implemented in two parts. The first part was to have the object tracking algorithm draw the estimated positions of the tracked humans as a small red box in each frame captured from the infrared camera (one box for each Kalman filter), and then storing this sequence of images as a *motion jpeg* (MJPEG) file locally on the PandaBoard. The second part was to implement a video streamer that could stream the MJPEG file over the *http protocol*. The reason for choosing this protocol is that Neptus has built in support for such video streams, which in turn makes it very easy to interface the video stream to the ground station. Interfacing a http video stream to Neptus can be done by simply connecting Neptus to the URL of the video stream. An illustration of how

Figure 5.5: The tracking algorithm writes the images that it processes (with marked object locations) to a file stored on the SBC. The video streamer reads from this file, and streams its content over the HTTP protocol. The control station views the stream by connecting to the correct HTTP address.

the video streaming module works is shown in figure 5.5. The video streamer was implemented by using the open source plugin *MJPEG Streamer* [3].

The data logger was implemented in a very straight forward manner. That is, for each image frame captured from the infrared camera, object tracking is performed as normal. Now, after the tracking algorithm has updated the Kalman filters of the objects that are being tracked, the estimated position of these objects are stored in their own respective trajectory vector. This continues as long as the objects are being tracked. When the target is lost, or every one hundred frame, the trajectory vector is written to a data file located on the SBC before it is reset. In the data file located on the SBC, the object ID (one ID per Kalman filter) and location in the image plane at each frame is stored. The frame number (counted incrementally and reset every 2^{31}th frame)[1] at each position is also stored, so that each position is marked with a 'time stamp'. A parser that is able to extract and display position trajectories of an object based on a given ID was implemented and used to compare real position versus estimated position of tracked objects. Note that the data logger should be coupled with information from the GPS, as an object's position in the image plane may provide limited information about the object. Consider the case where the UAV is tracking a boat that is moving in the same direction, and also with the exact same speed as the UAV. The boat can appear stationary in the image plane, when in reality it is moving very fast. Hence, interfacing the data logger to GPS data is important and should always be done when possible. However, this was not implemented in the present thesis.

The navigation algorithm was implemented only to illustrate a 'proof of concept', i.e the navigation algorithm was designed so that it outputs some navigational

[1]The reason behind this reset is that the frame number is stored locally as an *unsigned integer*. The maximum value of such a variable is 2^{32}, hence a reset is required when the frame number approaches this value.

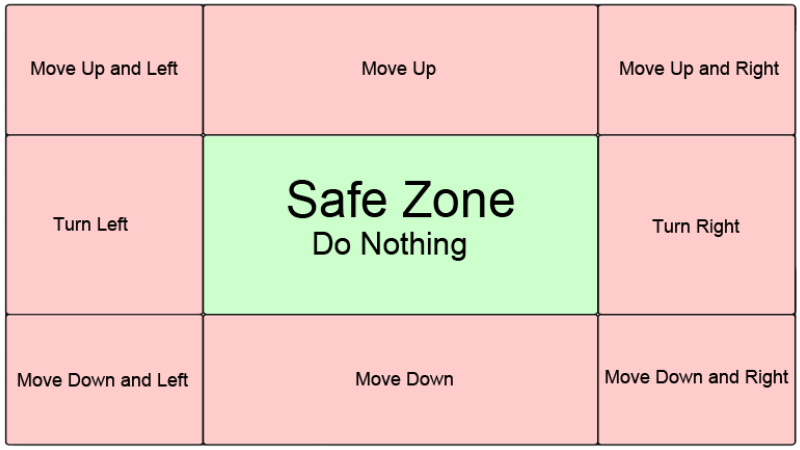

Figure 5.6: A very simple navigation algorithm. If the estimated location of a tracked object is within a red zone, the UAV should move according to the scheme illustrated.

commands based on the location of the tracked objects. These commands are then communicated to Dune. Now, Dune is not yet configured to do anything with these messages, hence the UAV is not actually controlled by this navigation algorithm. However, in future work, this is exactly how the object detection algorithm will take over control and navigation of the UAV, and implementing such an algorithm is therefore of great interest. The navigation algorithm implemented in the present thesis is based on a naïve approach. It divides the image plane into nine regions as illustrated in figure 5.6. Now, if the estimated position of a tracked object enters one of the red regions, a message is sent to Dune indicating that the UAV should move in order to keep the object within the field of view of the infrared camera. The navigation algorithm also visualizes this by drawing a small green circle in the critical region containing the tracked object on the video stream. Hence, the operator is also able to see what the navigation algorithm is communicating to Dune. The communication between the navigation algorithm and Dune is achieved by sending IMC data packets. In the case where Dune is running locally on the SBC (setup configuration 1), the messages are sent over the local memory bus, while they are sent over the local Ethernet network otherwise.

5.4 Field Testing

Having implemented and verified that the object detection and tracking algorithm, the video streaming module, the data logger and the naïve navigation algorithm was working as intended, it was of interest to evaluate the performance of the payload in a real life scenario. However, it was still considered important to be able to compare different choices for the parameters of the tracking algorithm from some sort of common ground. To satisfy these seamlessly contradicting interests, the real-time tracking was simulated on the PandaBoard. This was done by performing object detection and tracking on already captured video sequences, while at the same time keeping track the processing time of each image. Now, when a new frame was to be read from the video, the tracking algorithm was programmed to skip the amount of frames that it would have lost during object detection and tracking, given that the test was performed in a real-time scenario. In this way, the PandaBoard receives the exact same amount of frames and information as it would have done if the images were capture and loaded into the tracking algorithm in real-time. The number of frames that was skipped, was decided by the following equation

$$frames\ to\ skip\ = ceil(\frac{time\ elapsed\ since\ last\ frame\ capture}{frames\ per\ second\ in\ the\ video\ sequence})$$

where $ceil()$ is a function that rounds up to the nearest integer. It should be noted that the video streaming module, Dune and the data logger were all running locally on the PandaBoard as if the test was done in the field. By doing the tests in the described way, the real-time performance of the tracking algorithm could easily be evaluated for the different possible parameter choices for the tracking algorithm (e.g the Kalman filter parameters and type of classifier). Furthermore, comparing the choices made to each other could be done with in an unbiased manner.

All the tests were done by using a setup of the payload that was identical to setup configuration 1, with the exception of not having a GPS/INS device. After the payload was set up and functioning, its performance was evaluated by performing real-time single target tracking with the payload during the following situations

- Payload has no egomotion[2] and target is not moving.
- Payload has no egomotion while the target is moving.

[2]In computer vision, the term *egomotion* refers to the three dimensional movement of a camera within an environment.

- Payload has egomotion while the target is standing still.
- Payload has egomotion and the target is moving.

Before these tests were done, an optimal set of parameters were found by applying the real-time simulation to one of the already stored video sequences. The different parameters that were tested included the number of frames between in each full frame search (initially it was every 5 frames), the size of the search region created based on an object's estimated position and the Q, R and P_0 matrices for the Kalman filters. Benchmarks of the tracking algorithm such as image frames processed per second and tracking error were registered for each choice, and the settings that were found to yield best performance was used in the simulated field tests of the above mentioned scenarios. During these tests, the payload and the command center (external computer running Neptus) were connected to the same wireless network, and the command center was configured to display the video stream from the payload while communicating with the Dune client in the payload through IMC packages. This was done to verify that the connection between the two was working as intended. Before any of the simulations tests were performed, the software package developed for the payload was optimized for the PandaBoard. This was achieved by making sure that the SD card connected to the PandaBoard was of class 10[3] and configuring the operating system (Ubuntu) to only contain a bare minimum of extra features, e.g by removing the graphical user interface.

[3]The class of a SD card represent the speed at which it can be written to or read from. Class 10 is the highest class (i.e the highest read/write speed) available for these type of storage mediums.

Chapter 6

Results and Discussion

Based on the results gathered from the process described in the previous chapter, some important conclusions can be made. At this stage in the development process, it is still important to evaluate and question every aspect of the implementation. The payload designed in the present thesis is not a perfect solution, hence locating the areas which need improvement is crucial for the future development of the payload. Now, in order to find out how the payload can be improved, the following points were considered important

1. Evaluate the performance of the trained classifiers.
2. Evaluate which of the two object detection and recognition algorithms is best suited for application in the implemented object tracking algorithm.
3. Evaluate the performance of the tracking algorithm when applied to already stored video sequences.
4. Evaluate the performance of the total payload based on experience and data gathered from the field test.

The following chapter is based on the above mentioned list, and features a review and discussion of the most important results that were found.

6.1 Performance of the Classifiers

As mentioned in the previous chapter, a good way to evaluate the performance of a classifier, and also compare classifiers against each other, is by inspecting their ROC curve. The resulting ROC curves for the three different SVM/HOG classifiers are shown in figure 6.1, and for the three different BC/HL classifiers in figure 6.2. Note that the maximum TPR and FPR of a classifier (the point on the ROC curve located at the top right position), is equal to the case where $n = 0$. Hence, increasing n is equal to moving down and left on the ROC curves. n is still defined as the number of search windows needed to be classified as a human per actual human being classified. This means that $n = 0$ is equal to the case where there are *no requirements* on the amount of neighbouring search windows also needed to be classified as positive in order to return a region of interest as positive. It should be noted that even though the false alarm rate appears small for all of the classifiers, the number of search windows per image must be considered. E.g consider the case where an image of size $640x480$ is searched for objects of the size $64x128$. Letting the search window move 4 pixels at a time, the number of regions that has to be checked is equal to

$$(\frac{640-64}{4}+(\frac{64}{4}-1))*(\frac{480-128}{4}+(\frac{128}{4}-1))=18921$$

which means that even with a false alarm rate as small as 1.78×10^{-4} (the false alarm rate for the original SVM/HOG classifier at $n = 0$) there will be an average of ~ 3 false alarms *per image searched*. This performance can be considered sub par.

Now, looking at the ROC curves for the three different SVM/HOG classifiers, it is seen that all of them has a very high maximum true positive rate. This means that all of the classifiers are able to find most of the humans in video sequences that are similar to the test data. Furthermore, as seen from figure 6.1, training the SVM/HOG classifier using the mirrored version of the training set yields a better result. Further improvement can also be made by adding more negative examples to the training set, i.e it is observed that adding novel data to the negative examples noticeably decreases the false alarm rate for the classifier. Moreover, the positive effect of adding novel data is bigger than the effect of training with the mirrored data set included. However, increasing the size of the training set by adding only negative examples yields a poorer performance for classifying the positive examples, when compared to the performance of the other two classifiers. This can be seen from the fact that there is a small decrease in maximum TPR for the classifier trained using both the mirrored data set *and* the 500 extra negative

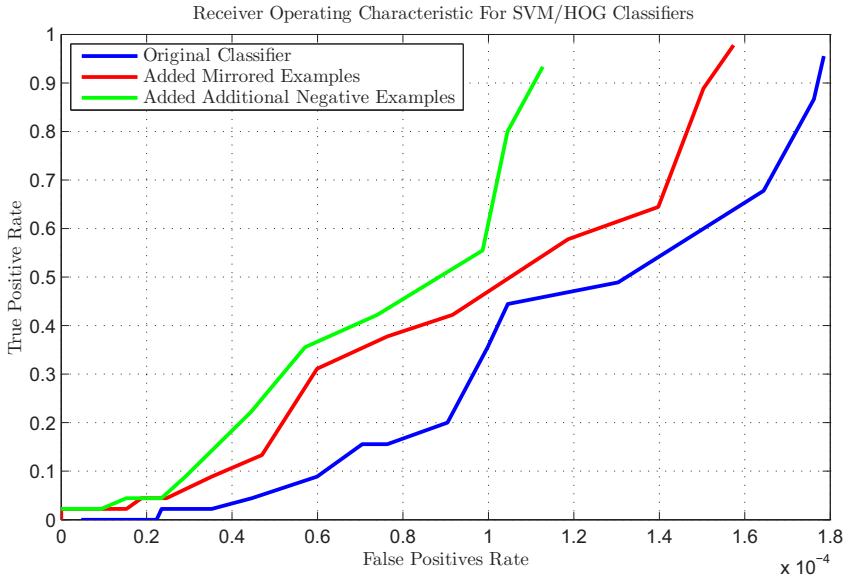

Figure 6.1: The ROC curves for the trained SVM/HOG classifiers. The required number of positive classifications in the vicinity of each other per actual positive classification was varied between $n = 0..20$.

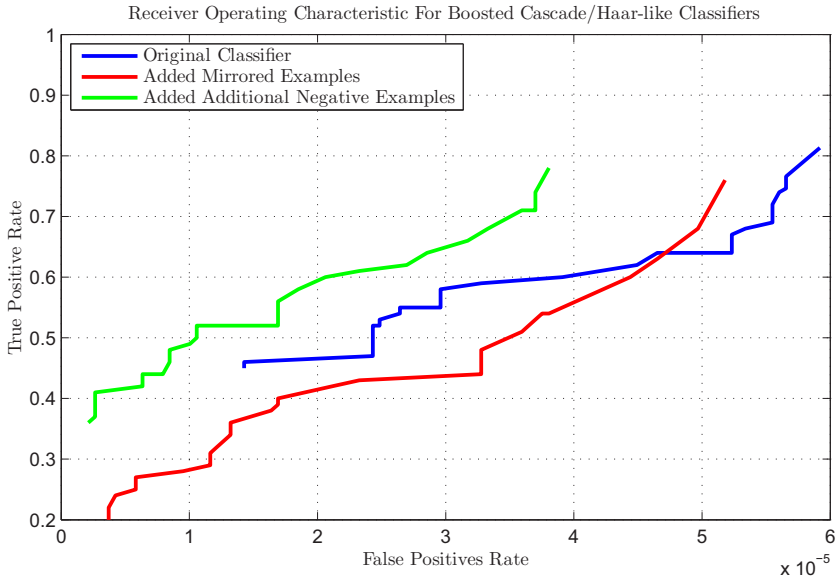

Figure 6.2: The ROC curves for the trained BC/HL classifiers. The required number of positive classifications in the vicinity of each other per actual positive classification was varied between $n = 0..20$.

examples. However, it is seen that the decrease in the TPR is small compared to the decrease in the FPR, which yields a net increase in performance.

For the BC/HL classifiers, a similar trend to that of the SVM/HOG classifiers is seen regarding the effect of increasing n. However, the BC/HL classifiers responds a little better to the increase of n, as the TPR remains relatively high (at least when compared to the SVM/HOG classifiers), while the FPR approaches zero. This gives more freedom and control when applying the classifier in the object tracking algorithm, as the number n can be varied until (hopefully) a sufficient tracking performance is achieved. Note that the maximum TPR *and* FPR of the BC/HL classifiers are overall markedly lower than the rate for the HOG/SVM classifiers.

It should be noted that even though the trained BC/HL classifiers has a significant edge in performance over the SVM/HOG classifiers, this should in no way be

seen as a generalization of the performance of the two. On the contrary, this was only an evaluation of the training process of the two classifiers, and a review of the amount of training data needed for each classifier to perform adequately well. Both types of classifiers are classifiers known to achieve remarkable results in e.g the task of pedestrian detection, with state-of-the-art classifiers of the type SVM/HOG actually performing *better* than the BC/HL classifier. However, it seems like this classifier requires a very large and varied training set to reach the point where it performs better than the BC/HL classifier.

6.2 Comparing the Classifiers

When different classifier types are to be compared, the first thing that has to be decided is *how* they should be compared. In the present thesis, these are the characteristics of the classifiers considered most important

- Training time
- Detection speed
- Performance on novel data
- Invariance to
  - Orientation/Rotation
  - Size
  - Changes in intensity

Now, the difference in the training time for the two types of classifiers used in the present thesis, is in the order of several days. In fact, the average training time for the SVM/HOG classifiers were only 20 minutes, while it was on average *three days* for the BC/HL classifiers, even though support for multiple CPU cores had been added using the *OpenMP* library[1]. The long training time is due to the extreme amount of different Haar-like features that it is possible to create within a $24x48$ pixels image. It should be noted that as the BC/HL classifiers are trained one stage at a time, it is possible to create a temporary classifier using only the

[1] *OpenMP* is an open source library which makes it easy to implement parallel computing independent of OS and CPU architecture

Table 6.1: The time used by different classifiers to perform object detection on a set of 1000 images with a resolution of $720x486$, moving the search window only 1 pixel at a time. The multi scale classifiers were set to detect object sizes up to ±50% the original training size ($64x128$)

Classifier Type	**Total Processing Time in Seconds**	**Frames per Second**
HOG/SVM Single Scale	216	4.6
BC/HL Single Scale	56	17.9
HOG/SVM Multi Scale	644	1.56
BC/HL Multi Scale	123	8.2

already trained stages for detection. However, it should be self-explanatory that the performance of these temporary classifiers may be very poor.

The detection speed of the classifiers is the rate at which the classifiers can be used to search an image for objects. Defining the rate to be measured as images per second, the time required by each classifier for performing detection on a set of 1000 images of the size $720x486$ can be seen in table 6.1[2]. It is clear from this table that the BC/HL classifier is noticeably faster at detecting than the SVM/HOG classifier. However, since the SVM/HOG has a higher TPR than the BC/HL classifier, the high TPR can be sacrificed for increased detection rate by increasing the distance the search window is moved between each detection, effectively reducing the number of search windows per image. This will increase the detection speed, but also decrease the TPR. The detection time for the SVM/HOG classifier can also be somewhat reduced by e.g reducing the number of overlapping cells per block, but this might also affect the performance of the classifier as the HOG descriptor will have fewer dimensions. This is also the case if the SVM/HOG classifier is trained on positive and negative examples of a smaller resolution.

When it comes to performance on novel data, it is seen from the ROC curves that the SVM/HOG classifier consistently has both a higher TPR *and* FPR for $n = 0$. However, when n is increased, the BC/HL classifiers has a higher TPR and lower FPR than the SVM/HOG classifiers. This means that which classifier is better to use really depends on the ability (or possibility) to remove false positives by some other means. In chapter 3 some techniques to filter out non interesting

[2]The computer used was a desktop computer with a quad core i5-2400 3GHz CPU

Table 6.2: Summary of the comparison of the two classifiers. More + and − signs indicates a larger difference.

Characteristic	**HOG/SVM Classifier**	**BC/HL Classifier**
Training Time	+ + +	− − −
Detection Speed	− − −	+ + +
Performance on Novel Data	+	−
Orientation Invariance	+	−
Size Invariance	−−	++
Intensity Invariance	++	−−

regions are proposed, and will in many cases give the SVM/HOG classifier a tiny advantage over the alternative. Furthermore, an aspect not considered in the test set, is the ability of the SVM/HOG classifier to detect humans with significantly different temperature than the temperature of the humans in the training set. As previously mentioned, the temperature of humans compared to the temperature of the environment might change their appearance in IR imaging completely, hence, good detection for both cases based on only one classifier may be difficult. However, the SVM/HOG classifier is partly invariant to this because it bases the detection on *contour* rather than *intensity values*. That is, even though the color of the humans change, their contour do not. This results in that a SVM/HOG classifier trained using images of relatively hot humans, can also be used in some cases to detect humans relatively cold. The BC/HL type of classifiers would have to be trained for both cases separately, effectively having one classifier for humans that are hot compared to the environment, and another classifier for humans that are cold compared to the environment.

Regarding the invariance with respect to factors such as rotation, size and intensity, the SVM/HOG classifier is the more robust alternative. This can be seen from the fact that this classifier is partly rotation invariant, as objects can be rotated $\pm\frac{180^\circ}{2\times number\ of\ bins}$ and the classifier will still classify correctly. This is because when the HOG descriptor is constructed, a rotation of the mentioned order will not affect which rotation bin the gradients are sorted into. The BC/HL classifier can be trained to tolerate small rotations by increasing the training set and including rotated examples of the positive training data. However, if the training set includes a big variety of object orientations, the classifier will not perform well as the HL features that discriminate correctly on one example is going to fail on another. The SVM/HOG classifier is also more robust to changes in the image intensity, as the

HOG descriptor is normalized within each block. This yields a good performance invariant of how bright the object appears, as long as its contours are visible. The BC/HL classifier will tolerate small changes in intensity, but if it changes markedly from the training set, the threshold set for each HL will no longer be appropriate.

The comparison of the two classifiers are summarized in table 6.2.

6.3 Performance of the Tracking Algorithm

The tracking algorithm was implemented as described in the previous chapter, and the performance of the algorithm given different choices for the design parameters was evaluated using several prerecorded video sequences. The performance was evaluated by establishing a ground truth of the location of humans in the video sequences, i.e manually registering their actual location in the image frames. The ground truth was then compared to the estimated position of the tracked humans. The estimated positions of the tracked targets was available through the implemented data logger module. The performance of the tracking algorithm was then evaluated by plotting the tracking error along with the tracking characteristics for each case in graphs such as figure 6.3 and 6.5. The absolute value of the tracking error at time step k was defined as follows

$$|\epsilon[k]| = \sqrt{(\mathbf{p}[k] - \hat{\mathbf{p}}_{k|k}[k])(\mathbf{p}[k] - \hat{\mathbf{p}}_{k|k}[k])^T} \tag{6.1}$$

where $\mathbf{p}[k]$ is the ground truth (manually registered position) at time step k, and $\hat{\mathbf{p}}_{k|k}[k]$ is the *a posteori* position estimate at time step k. This was an effective method to evaluate the performance of the tracking algorithm for different parameters, as it is easy to verify that a human is being correctly tracked in addition to that it makes it easy to compare tracking errors. It should also be noted that since the real position was made manually, the real position is inaccurate at some points. This results in that parts of the tracking error may be due to an inaccurate mouse click, rather than an error in estimated position. However, this error was considered to be small compared to the tracking errors, and was disregarded in the evaluation of the tracking performance. Furthermore, it was wanted to compare the use of the BC/HL classifier type for detection to the case of using the SVM/HOG classifier type instead, in a non biased way. This was achieved by applying already well trained classifiers for object detection during the tests that were done. Specifically, the applied BC/HL classifier was the one developed by Kruppa, Castrillion-Santana

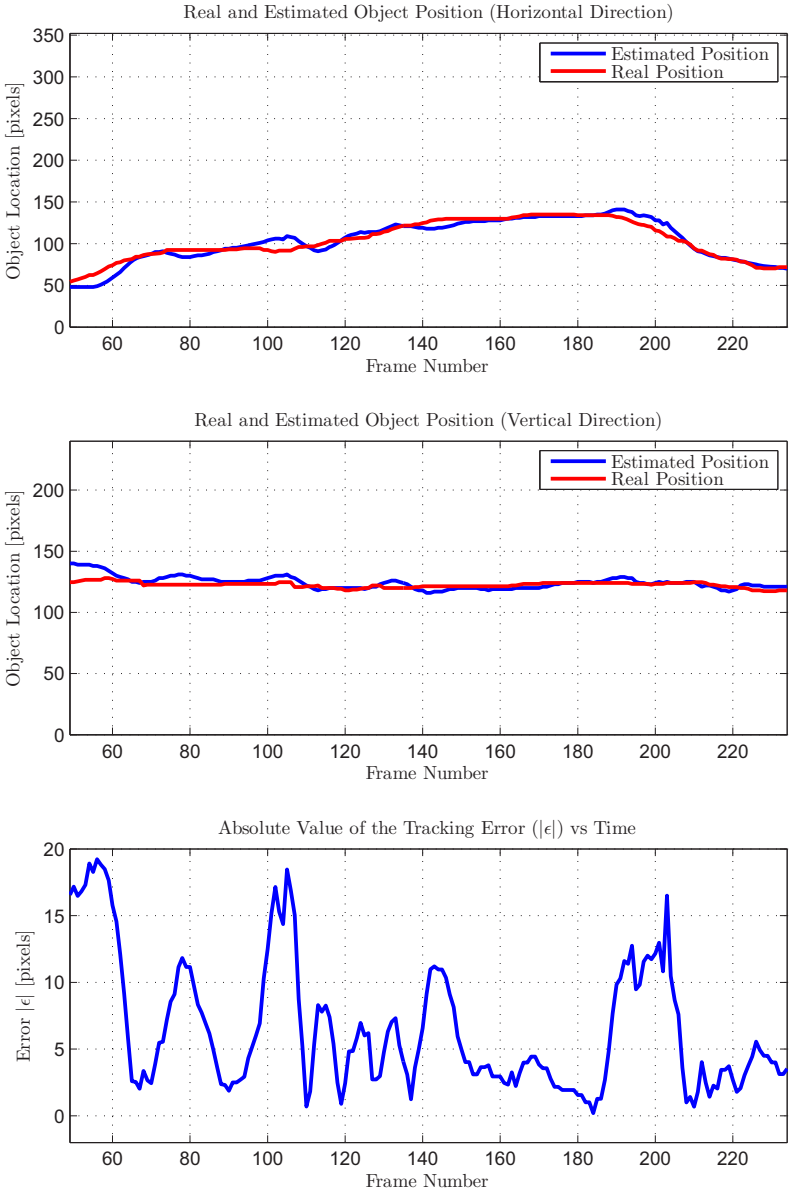

Figure 6.3: The tracking performance of the implemented tracking algorithm combined with a HOG/SVM classifier for the detection step. Note that the overall tracking achieves a very small tracking error, with small spikes in the error when the target changes velocity.

and Schiele [21], and the SVM/HOG classifier was the one originally developed by Navneet Dalal and Bill Triggs [14]. Both are classifiers known to perform well for human detection in thermal imaging.

Now, the tracking error for different choices for the parameters when tracking the same human was considered a reasonable method to compare the different choices. The tracking error of some of the most promising choices for the matrices Q, R and P_0 that were found are summarized in figure 6.4.

From this figure, it is seen that tracking with $Q = R = P = I$ gives relatively large spikes in the tracking error. This is due to the fact that with a large Q matrix, small differences in the measured position versus the estimated position will yield a relatively large change in the estimate for the velocity state in the motion model. Hence, in the absence of measurements, the Kalman filter will drift away from the target. Only when a new measurement is available is the estimated position "pulled back" to the real position. Since the R matrix is set equal to Q, this results in a Kalman gain (as seen in equation 3.7) which is large enough to make the process of correcting for a big difference between measured and estimated position happen almost instantly.

For the choice of $Q = R = I \times 10^{-2}$ and $P_0 = I$, Q and R are still equal, hence, a very similar tracking behavior is observed. The difference is that the tracking error now is in a smaller scale, and that the final spike in the tracking error for this choice of parameters is actually due to the Kalman filter changing which target it tracks. This happens because there is another human right next to the originally tracked human, and that there is only one measurement available for both of them at frames $\sim 185-190$. Since R now is smaller than in the first example, the tracking algorithm trusts the measurements more. This results in that the available measurement, which is closer to the human not being tracked, is quickly adapted as the real position. This results, as mentioned, in that there is a change in which human the Kalman filter is tracking.

The confusion of which human to track is also the reason for the large spike in the tracking error starting at frame 185 for the next choices of parameters, i.e $Q = I \times 10^{-2}$ and $R = P_0 = I$. For these values, Q is chosen relatively small compared to R and P, resulting in a smaller Kalman gain. This yields a smoother tracking characteristic compared to that of the previous choices. Furthermore, when the measurement that is matched to the Kalman filter is the measurement of the position of the second human instead of the tracked human, and in addition this happens several frames in a row, the Kalman filter adapts based on the wrong

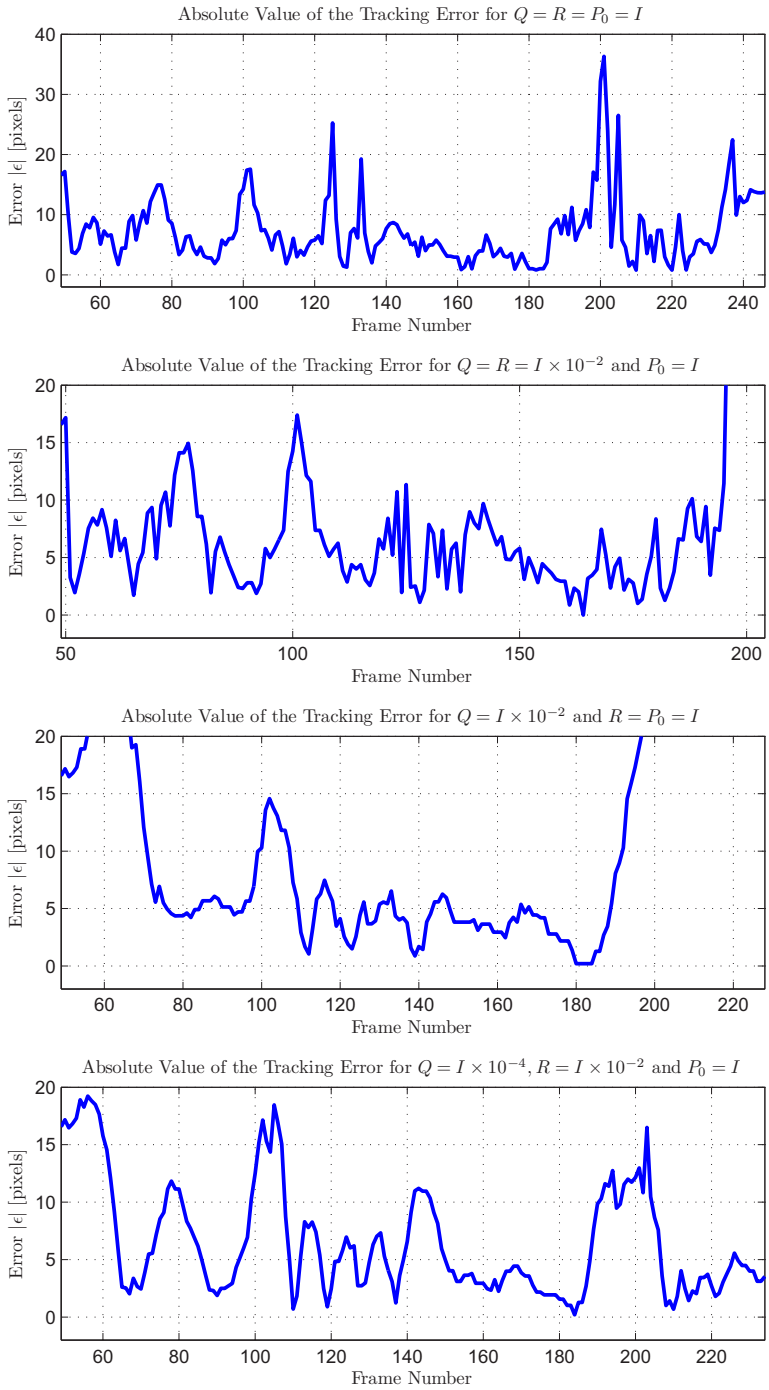

Figure 6.4: The tracking error for various choices of the Q, R and P_0 matrices of the Kalman filter. Note that all tracking errors are based on the same video sequence.

measurement. Hence, when measurements of the human actually being tracked are available again, it is too late to change the course of the estimated motion model states of the Kalman filter. This results in that the human being tracked changes also for this configuration.

Lastly, the parameters $Q = I \times 10^{-4}, R = I \times 10^{-2}$ and $P_0 = I$ was tested. Since Q now is chosen small compared to R, but R still is chosen relatively small, the tracking characteristic appears smooth, but not as smooth as for the case where $R = I$. However, as seen from figure 6.4, this appear to be a good combination of parameters, as the average tracking error is low, and the tracked human is neither lost nor confused with the second human in the vicinity of the one being tracked. This means that these parameters results in a Kalman filter that is well suited for tracking objects in scenarios similar to the one tested. Hence, the matrices Q, R and P_0 were chosen as

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad Q = \begin{bmatrix} 1 \times 10^{-4} & 0 & 0 & 0 \\ 0 & 1 \times 10^{-4} & 0 & 0 \\ 0 & 0 & 1 \times 10^{-4} & 0 \\ 0 & 0 & 0 & 1 \times 10^{-4} \end{bmatrix} \qquad (6.2)$$

$$R = \begin{bmatrix} 1 \times 10^{-2} & 0 \\ 0 & 1 \times 10^{-2} \end{bmatrix}$$

Having made a choice for these parameters, it should be emphasized that there probably is no universal best set of parameters. This is because the best choice depends on the velocity and motion of the targets being tracked. In most of the video sequences that were recorded, the humans are not moving especially fast by the means of pixels per frame. More importantly, they were moving at very similar speeds (if moving at all), hence the parameters should be subject to change when applied to different scenarios. Furthermore, considering the motion model described in equation 3.3, it is reasonable to model the process noise as an ever present acceleration a_k which is normally distributed with $E[a_k] = 0$ and standard deviation of σ_a, i.e Gaussian white noise. Now, with this assumption, we can rewrite the linear model using Newton's laws of motion such that it takes the following form

$$\mathbf{x[k+1]} = \begin{bmatrix} x[k+1] \\ y[k+1] \\ v_x[k+1] \\ v_y x[k+1] \end{bmatrix} = \begin{bmatrix} x[k] + \Delta T v_x[k] + \frac{1}{2}\Delta T^2 a_k \\ y[k] + \Delta T v_y[k] + \frac{1}{2}\Delta T^2 a_k \\ v_x[k] + \Delta T a_k \\ v_y[k] + \Delta T a_k \end{bmatrix} = A\mathbf{x}[k] + K a_k$$

where K is the matrix

$$K = \begin{bmatrix} \frac{\Delta T^2}{2} \\ \frac{\Delta T^2}{2} \\ \Delta T \\ \Delta T \end{bmatrix}$$

Noting that Ka_k is the term $\mathbf{w_k}$ in the linear motion model derived in equation 3.3, together with the fact that the matrix Q for the Kalman filter is equal to $cov(\mathbf{w_k}, \mathbf{w_j})$ (equation 3.7), the result is the following

$$Q = cov(\mathbf{w_k}, \mathbf{w_j}) = (K\sigma_a)(K\sigma_a)^T = \begin{bmatrix} \frac{\Delta T^4}{4} & \frac{\Delta T^2}{4} & \frac{\Delta T^2}{2} & \frac{\Delta T^2}{2} \\ \frac{\Delta T^2}{4} & \frac{\Delta T^2}{4} & \frac{\Delta T^2}{2} & \frac{\Delta T^2}{2} \\ \frac{\Delta T^2}{2} & \frac{\Delta T^2}{2} & \Delta T & \Delta T \\ \frac{\Delta T^2}{2} & \frac{\Delta T^2}{2} & \Delta T & \Delta T \end{bmatrix} \times \sigma_a^2 \tag{6.3}$$

which strongly suggest not only that the process variance of the states are dependent on each other, but that the entries of Q should be scaled accordingly to this result. Tests based on this result were not researched to any significant extent, but should be considered in the future when searching for good values for Q.

Having established some settings for the Kalman filter that yields good tracking characteristics, and also a low tracking error, the overall performance of the tracking algorithm must be evaluated. From figure 6.3 it is seen that the tracking error starts off relatively high. This is reasonable, as the Kalman filters are initiated with initial velocity 0, but the human is actually moving. This means that the estimated states of the motion model initially are wrong. Because of this, the quality of the prediction step is poor, and this results in a relatively high tracking error. It is seen that after ~ 20 frames, the measurements have changed the estimated states, and the predicted position is almost equal to the real position. The next spike in the tracking error (at frame ~ 110) was due to an occlusion, and was the result of tracking without any available measurements. It is readily seen that the estimated position in frames 90 – 110 is the result of the Kalman filter assuming that the human continues to move at the same speed as the estimated velocity at the time of the last measurement. However, the choice made for the values in Q keeps the estimated position in the vicinity of the real position, and when a measurement becomes available at frame ~ 110, the estimated states are corrected. The final spike in the tracking error (at ~ 200) is a good example of the negative sides of choosing a relatively small Q. Since the estimated velocity is ~ 0, the estimated position lags behind the real position when the human suddenly starts to move. This illustrates that choosing Q is a matter of balancing good predictions and adaptability. In this example, the tracking error spikes at

110 and 210 are almost equal, which means that the chosen Q was a decent choice.

Looking at figure 6.5, another aspect of the performance of the tracking algorithm is illustrated. Initially, at frames 910 – 940 there are two humans in the image frames, but the distance between them is so small that both of these measurements are considered to be associated with the same Kalman filter. This results in that only one of the humans is being tracked. However, when the tracked human moves a little bit further away from the other human in the image, the Kalman filter manager realizes that there are actually two humans present. This results in that another Kalman filter is initiated, and tracking of both humans is commenced. Notice that even though tracking two humans standing next to each other is no extraordinary feat, it must be specified that for this to be possible, it is a requirement that there is no mix up of the two measurements. This illustrates that the implementation of the GNN matching algorithm was successful. Note that a NNSF probably would have failed at this task, as both measurements are within the vicinity of each other, resulting in the occasional mix up described in figure 3.8.

The final parameters to evaluate for the tracking algorithm is both the size of the search regions, and at which point on the ROC curve the classifiers should operate at. By varying these parameters and performing tracking on the video sequences, it was found that a high TPR, i.e $n = 0$ (at the cost of a relatively high FPR), combined with a relatively large search region ($\pm \sim 1 \times object\ size$), yielded good tracking performance, but also a significant number of Kalman filters initiated based on false positives. Most of these were quickly removed due to the lack of continued measurements, but in some cases the tracking based on these measurements could lasts for as much as up to 20 – 30 frames. Hence, for this to work properly, some other means of weeding out false positives must be implemented. A simple approach where the search region in the thresholded image was required to contain at least 40% high intensity pixels was implemented and verified to reduce the number of false positives. However, this requires extra computational power, and should therefore be avoided if possible. An alternative solution is to decrease the size of the search region created based on the predicted object location, but this is usually not a very good idea. Decreasing this region should generally be avoided, since the predicted location may differ from the real position by up to 20 – 30 pixels. Hence, decreasing the size of the search region increases the chance of not capturing the region where the tracked human is present. Now, increasing n, effectively moving the operating point of the classifiers downwards and left on the ROC curves, proved to increase the performance of the tracking algorithm. This is reasonable, because as the Kalman filter manager only removes or creates a Kalman filter every 5 frames, *one* measurement of an object within these 5 frames

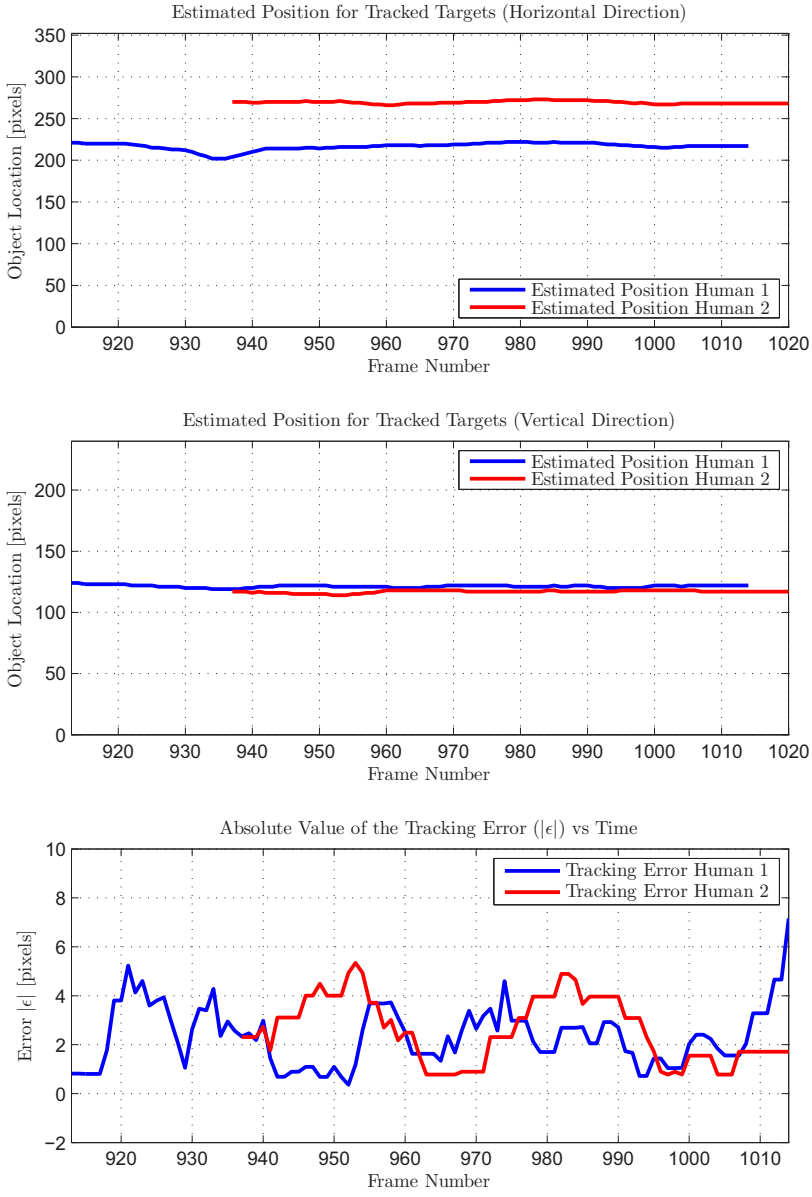

Figure 6.5: A test scenario where there are two humans present in the video sequence. At the beginning, the distance between them is so small that the two measurements are associated with the same Kalman filter. However, when the distance between them increases, the tracking of the second humand is also initiated. After both Kalman filters are initiated, the tracking algorithm manages to track both humans by correctly matching the measurements to the Kalman filters. This is achieved by using the global nearest neighbor approach.

is enough for the tracking algorithm to keep tracking the object. Hence, a low FPR is more important than to have a TPR in the region of ~ 0.9. Note that increasing n is only beneficial up until some point, because with a too low TPR, tracking will be lost and re-initiated from time to time due to the lack of measurements. For the well trained classifiers used during these tests, the optimal value for n was found to be 1 for both classifiers.

A final note regarding the performance of the tracking algorithm is the performance of the process of normalizing, thresholding and extracting components that has the shape of a human. This was found to be a process that required some tuning before functioning properly. That is, there was no universal threshold and normalization factor that yielded optimal detection of human blobs in all of the video sequences tested. This resulted in that these parameters had to be tuned for each video sequence to achieve an acceptable performance, which is not ideal, as the payload ideally should be able to perform tracking without interaction of an operator tuning the parameters. An alternative method to extract promising regions is to base the search on *contours* rather than intensity, as this is more robust to intensity changes. However, this approach was not tested. Furthermore, normalization was found to increase the detection rate when it was applied to *every* frame, and not only every fifth frame, i.e when the full image is searched.

6.4 Performance of the Total System

With the tracking algorithm implemented and the classifiers trained, it was of significant interest to test the payload in a real life, real-time scenario, as this is more like the scenarios it is intended to be applied to. However, interfacing the infrared camera to the SBC proved to be a big challenge. The frame grabber used in setup configuration 1 of the payload (EasyCap), has a community created driver for the Ubuntu operating system, but requires a Linux kernel version not currently available for the PandaBoard. Now, there exists software for the PandaBoard that enables the user to capture the *raw data stream*[3] coming from the infrared camera with the EasyCap device, hence it is not a problem to record thermal imaging videos through the use of this frame grabber. The problem occurs when the images are to be interfaced to *OpenCV*. OpenCV requires the frame grabber to be

[3]The raw data stream refers to the uncompressed and non-encoded data coming directly from the EasyCap device, containing metadata that enables the user to encode the raw data stream into a reasonable format, e.g an *.avi* movie file in this case.

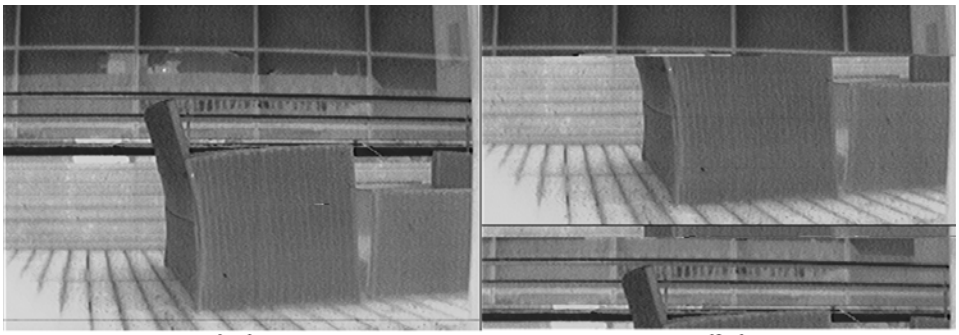

Figure 6.6: (a) illustrates an example where the implemented image capture library has successfully captured an image from the raw video stream, and has loaded it into the data structure used to store images in OpenCV. (b) illustrates that there are occasional synchronization problems between the image capture library and the raw video signal, resulting in a heavily occluded image.

registered as a capturing device in the operating system, which requires the drivers for the capturing device to be installed. However, as mentioned, the driver was not supported on the PandaBoard. This resulted in that the standard image reader functions in OpenCV was rendered useless. Hence. an implementation of a small library able to read the raw video, convert it to gray scale images and load it into the data structure used to store images in OpenCV was written in *C++*. The result from this work is seen in figure 6.6. This figure illustrates that the implemented library has some occasional problems with synchronization of the raw video signal, resulting in that some frames are heavily occluded. As no documentation for the software extracting the raw video stream from the EasyCap was available, the implementation of the image reader library was not an easy task, and does not work perfectly. It should be noted that further studies of the raw video signal coming from the EasyCap software will solve this problem.

Now, before the simulation of the real-time tests described in section 5.4 could commence, a set of parameters for the tracking algorithm yielding a good real-time tracking performance had to be found. The first parameter that had to be decided was which type of classifier that should be applied for optimal performance. To answer this question, a real-time simulation test of the two classifiers was performed. The result of this simulation is given in figure 6.8, and it is readily seen that apply-

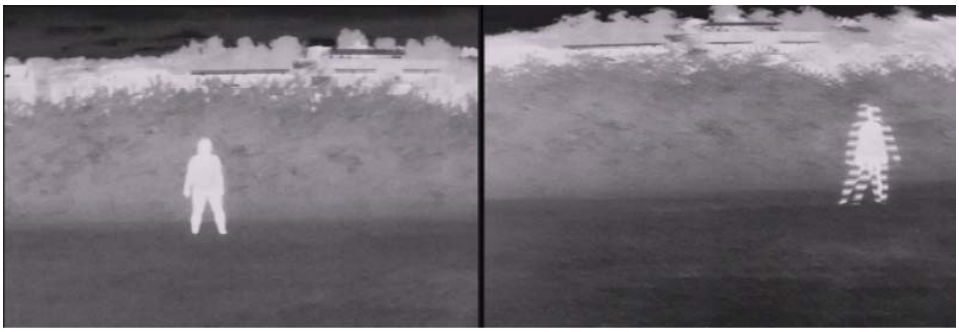

Figure 6.7: (a) shows a case where it is not possible to see that the image consists of two different parts, while (b) shows an illustration of interlacing effects. Since the target in (b) is moving relatively fast, the two parts that the image consists of are describing two different scenes, resulting in that the image countour of the human is occluded.

ing the BC/HL classifier in the detection step of the tracking yields a lower tracking error than using the SVM/HOG classifier. Now, the explanation for this is divided in two parts. Firstly, the detection time for the SVM/HOG classifier is much larger than the detection time for the alternative (BC/HL classification). This results in that the tracked target is able to move more during the time required for the SVM/HOG classifier to perform object detection. This means that the distance between the actual target location and the estimated target location may become relatively large in each processed frame. Furthermore, since the detection time for the SVM/HOG classifier is larger than that of the BC/HL classifier, fewer measurements are obtained per second, which yields a longer transition period. That is, the estimated target location does not converge as fast to the real position when using the SVM/HOG classifier compared to using the BC/HL classifier. This is mainly due to the difference in measurement rate that occurs because of the difference in the time used to detect objects. Now, the other part that plays a role in the observed tracking error is a concept referred to as *interlaced video*. Interlacing is the concept of doubling the perceived frame rate by dividing an image into the data contained in every even pixel, and the data contained in every odd pixel. These two parts are then updated only one at a time, meaning that only half of the pixels in an image needs to be transfered to the viewing application before a new frame is created. As mentioned, this increases the perceived frame rate, as a new image

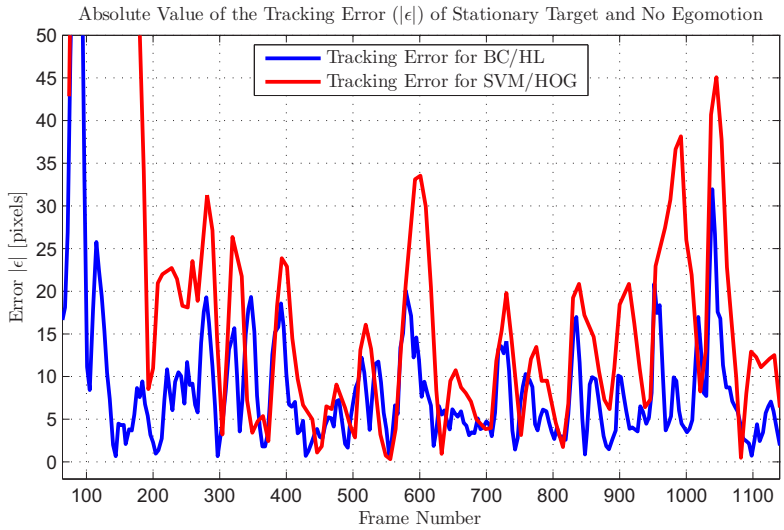

Figure 6.8: The real-time simulation tracking error for the two types of classifiers used in the present thesis. Note that the tracking error when using the HOG/SVM classifier for detection is on average larger than the tracking error when the BC/HL classifier is used for detection instead.

can be created by only exchanging *half of the information* otherwise necessary, i.e the case where every pixel in the image must be transferred before a new frame is created. The latter case is referred to as *progressive video*. Now, the NTSC signal coming from the infrared camera is an interlaced video signal, which is not ideal for the object detection and tracking algorithms. This is because the two different parts in the interlaced video are not taken at exactly the same time, which in turn introduces problems referred to as *interlacing effects*. An illustration of how this effect is perceived in an image is seen in figure 6.7. The interlacing effects occur when there is relatively large movements in the image because, under this circumstance, the two parts that make up the whole image has registered two different scenes. The interlacing effects can be seen to affect the *contour* of the target more than the intensity. Hence, the interlacing effects affects the SVM/HOG classifier more than it affects the BC/HL classifier, as the HOG descriptor is built from the contours of an object. This results in that the detection on frames with the interlacing effects will be poor when the SVM/HOG classifier is applied, resulting in an even bigger difference in the rate of the measurements coming from the SVM/HOG

classifier compared to the BC/HL classifier. It should be noted that the interlacing effects can be greatly reduced by a process called *deinterlacing*. This process requires additional computation power, which is already scarce in the PandaBoard, hence it was not tested. The Axis M7001 frame grabber proposed for application in setup configuration 2 actually has an internal deinterlacing function, and should therefore be used in the payload when possible. Now, to make sure that the difference in the tracking error found for the two classifiers was not *only* due to the interlacing effects, a real-time simulation test was done both on the PandaBoard *and* on a desktop computer on a video sequence containing interlacing effects. The result from this test is given in figure 6.9. From the tracking result in this test, it is revealed that even with the interlacing effects present, the SVM/HOG classifier *can* be used to successfully track humans in real-time. This implies that the biggest obstacle of using the SVM/HOG classifier for the detection step in target tracking during relatively large movement is the required amount of time to perform detection on the PandaBoard. In tests performed, the PandaBoard processed an average of 3.4 frames per second with the use of the SVM/HOG classifier in the detection step. Now, this is a reasonable amount of frames processed per second, but when interlacing effects are present, this may become a problem. If the classifier fails to classify a human in about half of all cases due to interlacing effects, the net supply of measurements are in the range of ~ 1 measurement per second. This is, as the results indicate, too low to perform efficient object tracking in the presence of large displacements in the actual object location. However, in situations where it is known that the target that is being tracked is slow moving in terms of pixels per second, the use of the SVM/HOG classifier should be encouraged. This is because even though the tracking error is noticeably larger than when the BC/HL classifier is used, the ability to detect the target under a large variety of circumstances (i.e changes in intensities) outweighs the importance of having a $\sim 10\%$ smaller tracking error. In addition to this, the interlacing effect in such cases is small.

Having established that the BC/HL classifier type performs better than the SVM/HOG classifier in scenarios similar to those described in section 5.4, this was the classifier type that was used in the remainder of the simulated tests. Now, to finally test the performance of the payload in simulated real-time scenarios, the payload was setup according to configuration 1 (minus the GPS/INS) and verified to function properly. Then, video sequences according to the scenarios described in 5.4 were recorded. The payload did not malfunction even once during video recording and simulated real-time tracking. This is a good indicator of that the PandaBoard received a stable 5V input, in addition to receiving a sufficiently large electrical current. The infrared camera also worked flawlessly, supplying the frame grabber with a continuous stream of images with the exception of small occasional freezes

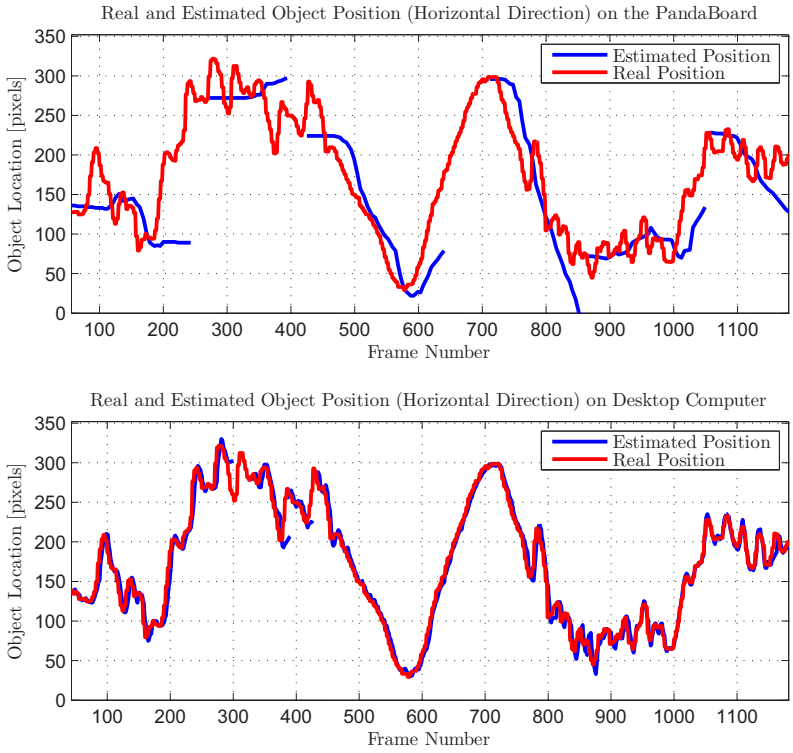

Figure 6.9: A simulated real-time tracking test with the SVM/HOG classifier used for the detection step. Notice that when the tracking algorithm is running on a desktop computer, the object location is tracked almost perfectly, while the tracking algorithm fails when it is used with the PandaBoard. This illustrates the importance of a regular supply of measurements to achieve good target tracking.

due to the camera refocusing the lens. Having verified that the payload was working as intended in addition to having recorded the necessary video sequences, the real-time simulation tests was performed. To evaluate the performance of different parameter choices, the results from the simulation tests are presented in a similar way to the results in the previous section. Different settings for the PandaBoard and the tracking algorithm was tested and compared. The settings for the Kalman filter was based on the results found in the previous section, and the optimal performance was found to be at values where the relative order between the matrices are similar to the optimal result found in section 6.3. That is, the following Kalman filter parameters were chosen as

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad Q = \begin{bmatrix} 1 \times 10^{-2} & 0 & 0 & 0 \\ 0 & 1 \times 10^{-2} & 0 & 0 \\ 0 & 0 & 1 \times 10^{-2} & 0 \\ 0 & 0 & 0 & 1 \times 10^{-2} \end{bmatrix} \qquad R = \begin{bmatrix} 1 \times 10^{-1} & 0 \\ 0 & 1 \times 10^{-1} \end{bmatrix} \tag{6.4}$$

The reason for increasing Q is that in the simulated real-time tracking case, the actual position of the tracked object changes more per measurement than in the case where each frame in the video sequence was processed. This means that the prediction step should include more insecurities, as moving too much in one direction may easily lead to that the tracking of target is lost. Furthermore, the increase in the measurement noise is due to the fact that even though the target is observed at some location, this may very well be a terrible estimate for the object position only one or two frames after the observation was done. Hence, care should be taken not to update the states of the motion model too fast. Increasing R results in a smaller Kalman gain, which in turn yields slower convergence to the measured state.

Having chosen some values for the Kalman filter parameters, the performance of the tracking algorithm was evaluated for several real-time scenarios. The tracking error of a stationary target and no egomotion of the camera can be seen in figure 6.11. For this test, the search region that is created from the estimated object location was made unnecessarily large. It was set to include 2.5 times the width of the object, and 2 times the height of the object. This was done both to enable the tracking algorithm to handle large displacements in the actual object location, but also to evaluate the computational power of the PandaBoard. A large search window yields a more computationally heavy tracking algorithm, which in turn can result in fewer processed frames per second. It should be noted that since the video sequences were recorded simply by holding the infrared camera in the hand,

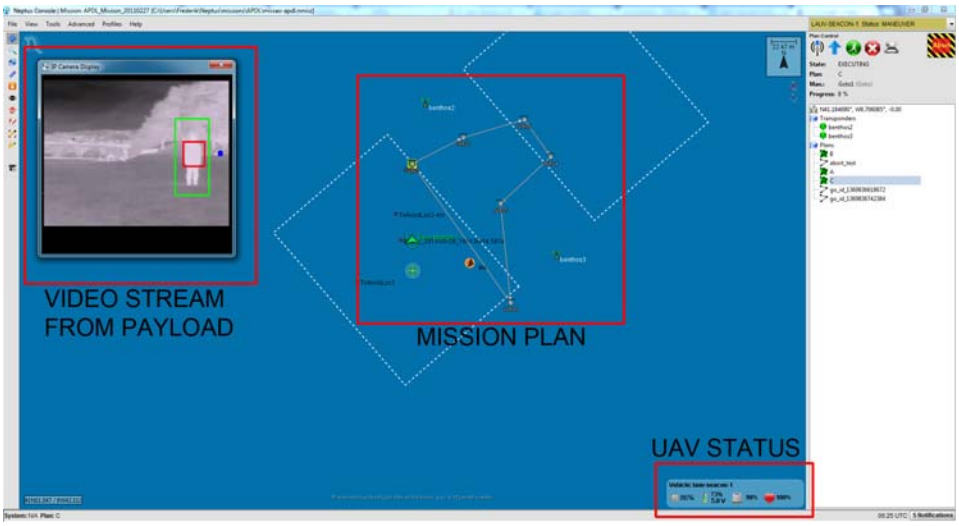

Figure 6.10: This screen shot is taken during a simulated real-time tracking test. The video stream sent from the payload is interfaced through a HTTP-streaming module in Neptus. The small blue circle located in the far right of the streamed video, is the visual queue that the navigation algorithm draws on every processed image when it wants the payload to move. The UAV status is showing CPU load, battery life and free storage space of the payload. The navigation control to make the UAV follow the path laid out in the mission plan was simulated and performed by Dune while the payload was performing object tracking.

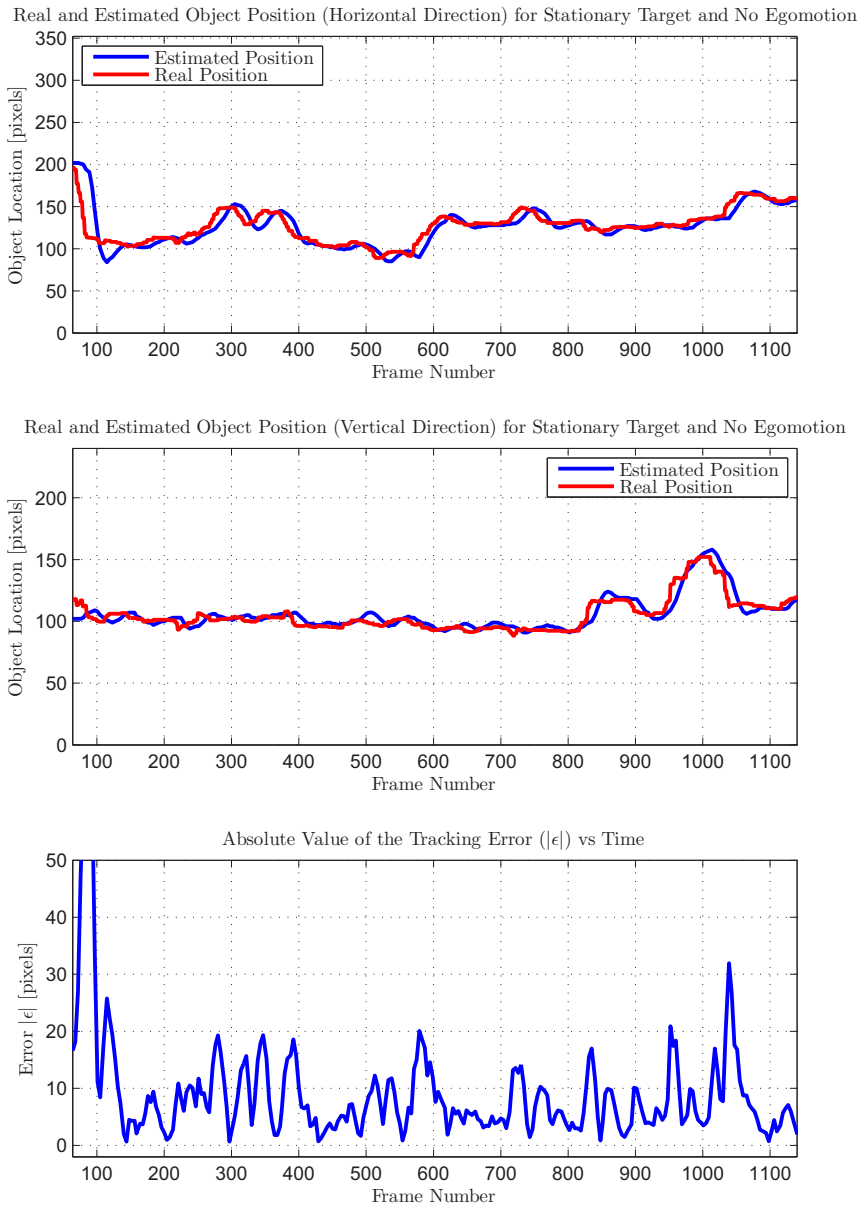

Figure 6.11: Simulated real-time tracking of a stationary target and no egomotion of the camera. It is observed that even though the estimated position lags behind the real position by some pixels, the tracking of the target is successful through the whole video sequence.

this introduced some vibrations to the location of the target. This is the reason for the appeared movement of the target by looking at the graph for the real position. These apparent vibrations are, however, not a problem for the tracking algorithm. The target is being tracked through the whole video sequence, making the test successful. In addition to this, the number of Kalman filters initiated based on false positives were *zero*. The video stream and communication with the ground station/Neptus was also working as intended during this test, and a screen shot of how this was observed can be seen in figure 6.10. It was also interesting to see that the tracking algorithm running on the PandaBoard processed frames at an average rate of 7.2 frames per second (FPS). This is, of course, due to the fact the whole image is only searched every 5 frames, but is still a remarkable detection speed. As seen in appendix D, the FLIR Tau2 IR camera has an upper limit on 7.5 FPS on cameras exported out of the United States. This means that even though the signal is up-sampled to 30 fps, the tracking algorithm makes use of almost 100% of the images taken from the camera. This means that increasing the rate at which the tracking algorithm operates will normally not increase the performance of the tracking by any significant amount. However, it should be emphasized that this was the *average* detection speed. As illustrated in chapter 5, the whole image is searched for new interesting regions every fifth frame. This search is more resource demanding than the intermediate frames where only a small part of the image is searched. Hence, this means that the object may move a large distance during this time, resulting in a large tracking error in spite of an average processing rate of 7.2 FPS. Finally it should be noted that the simulated real-time tests were performed using the Tau2 336 camera, which has a lower resolution than the Tau2 640. This means that the processed image rate for the latter IR camera will be a bit lower.

Moving on to the next test scenario, the simulated real-time test for the case where the object is moving and the camera has no egomotion is illustrated in figure 6.12. It is readily seen that also this test can be regarded as successful in terms of being able to track the target throughout the whole video sequence. However, it is seen that the estimated position lags behind the real position, especially in the situations where the target changes direction. This is to be expected, as the estimator bases its estimate on a linear motion model, while a sudden change in velocity is a very non linear process. Furthermore, some of the tracking error can be attributed to that the detection algorithm had some problems with detecting the human while he was in the process of making a step, effectively staying in the vicinity of the last observed location until the step was completed and a new measurement was supplied to the tracking algorithm. This did not lead to especially large tracking errors, and since the tracking was successful this was considered unproblematic. As a final note, it is seen that the human is moving from horizontal pixel value 50 to

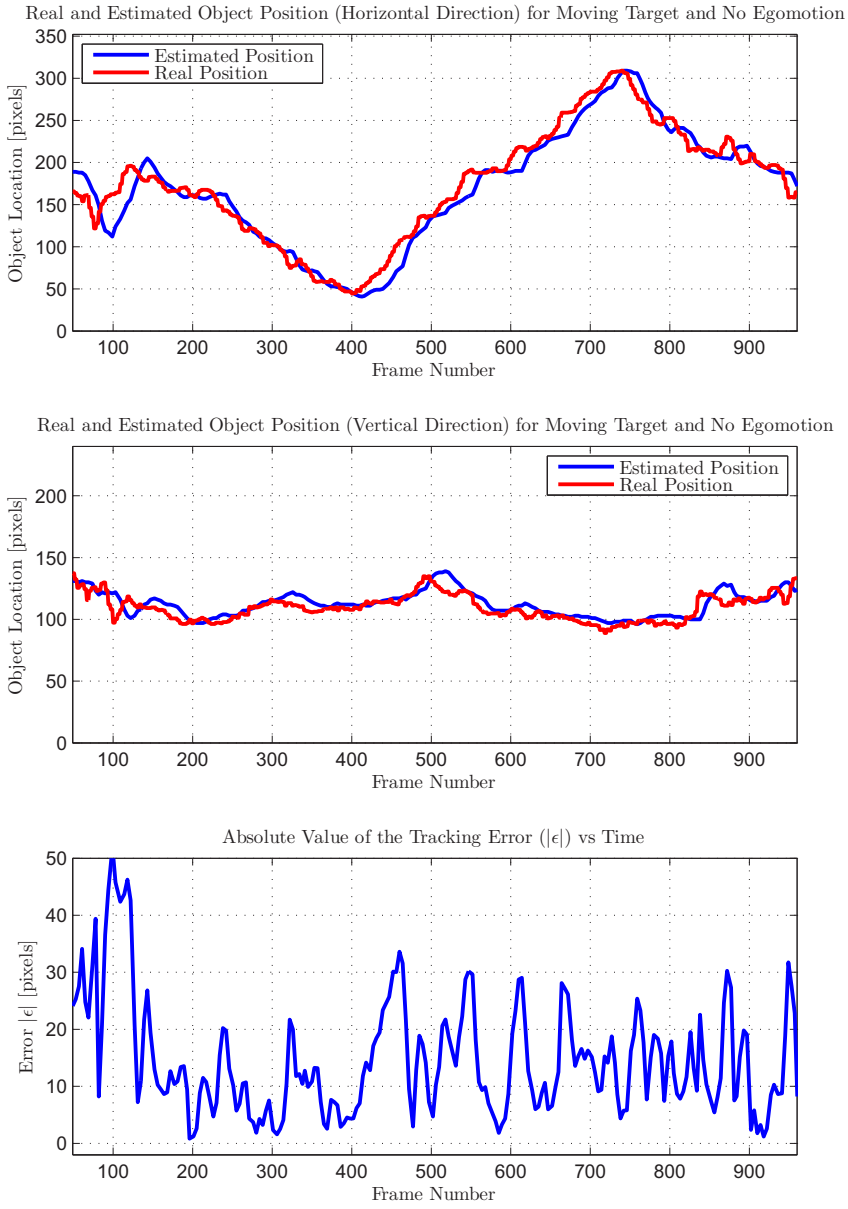

Figure 6.12: The result from the simulated real-time tracking test where the tracked target is moving and no egomotion of the camera. It is seen that the results are very similar to that of the previous stationary target tracking, with a small increase in the average tracking error.

a pixel value of 300 in the matter of ~ 340 frames. With a video sequence that has 30 FPS, this corresponds to $\frac{300-50\,pixels}{\frac{340}{30}\,seconds} = 22 pixels/second$, which again is equal to a rate of $\frac{22}{350} = 6.3\%/second$. Being able to track a target moving at such speeds is only possible due to the high amount of processed images per second, and the HOG/SVM classifier would probably have failed at this stage.

The next test was a bit more interesting. This test was based on a video sequence where there is relatively large egomotion of the camera. In addition to this, the video sequence was recorded while walking, introducing a small high frequency change in the actual position of the target, in addition to the large slow frequency movement due to the camera actually moving about in an environment. From this test, it is not as clear as for the previous examples whether the test was a success or not. It is observed that for ~ 90% of the time, the target tracking is more or less correctly tracking the target. However, the problem arises when the camera makes an abrupt and quick change in position. This is seen at frame 200, and then again on frame 1000. It should be emphasized that not only does the object tracking cease to track the target, but maybe more importantly, a re-initialization of the tracking is not achieved until the target more or less appears stationary. This is due to the fact that the whole image is only searched every fifth frame, hence the tracking algorithm only gets the chance to re-initiate the tracking once every fifth frame. Now, the classifier correctly classifies the human as a human at all almost every opportunity in the frames 200-300 and 1000-1100, but since the egomotion of the camera is large, the search region created by the predicted position of the target (which is a rather poor prediction, since P is initiated large) does not contain the target, hence the tracking algorithm does not receive confirmation that there actually is a human present there. This may be fixed in a number of ways, where one solution would be to simply enlarge the search window created based on the estimated position. However, this is not a very efficient approach, and the best solution is probably some sort of adaptive algorithm. That is, an algorithm that makes the search window initially large, and then reduces the size of this window as the certainty of the predicted estimate increases. Furthermore, when no target is being tracked, every frame can be fully searched (instead of every fifth), effectively increasing the chance of finding and commencing tracking of a target if it is present in the image plane. These are things that was not tested, and the size of the search window was not increased any further, as it was already considered large enough (2.5×width and 2×height of the object) and increasing it further beats the purpose of using the search window in the first place. The focus should be to develop a good and efficient way to avoid this problem. Now, during the rest of the tracking, it is seen that with the exception of a drift of the estimated position at

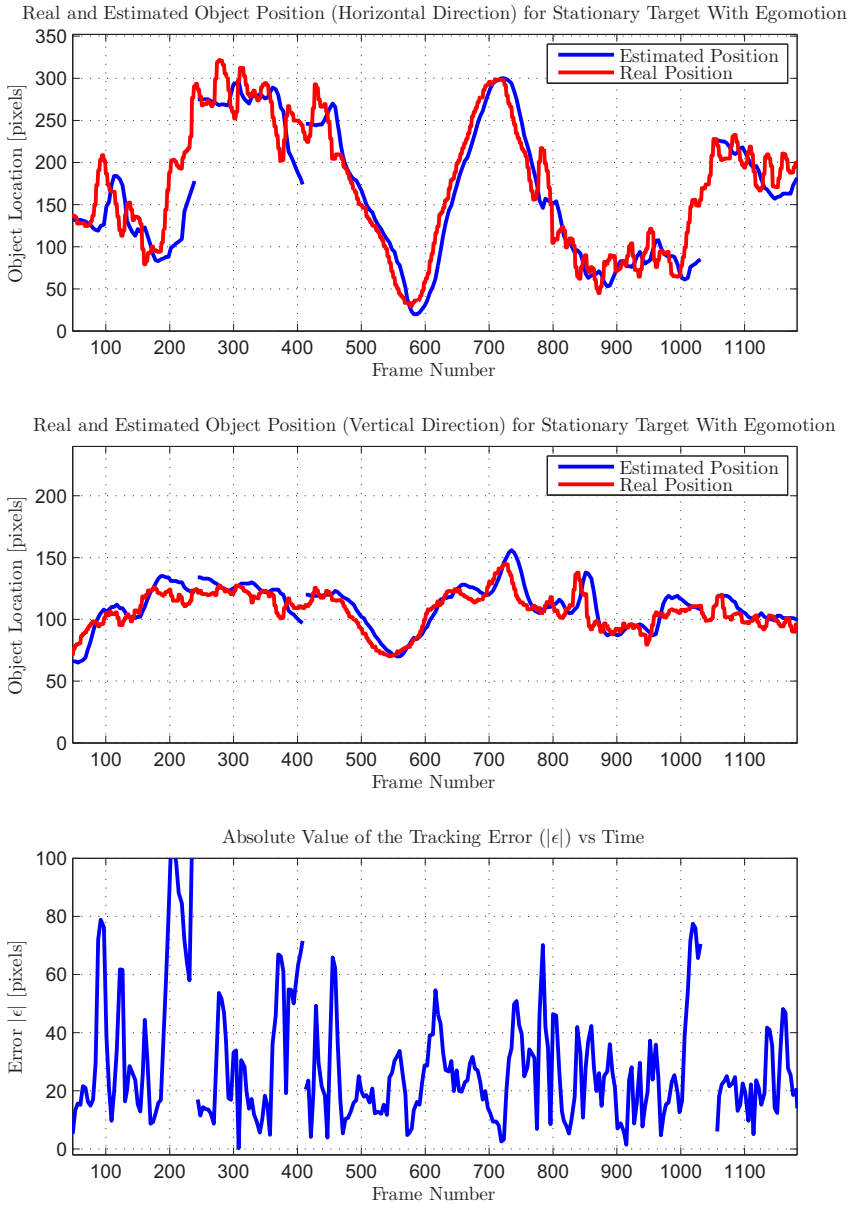

Figure 6.13: Simulated real-time tracking with egomotion of the camera and a stationary target. It is seen that the overall tracking performance is good, with the exception of tracking when the egomotion of the camera makes an abrupt and large change. This is considered reasonable as the estimator is based on a linear motion model.

frames $\sim 390 - 400$, the overall tracking performance is good. Most impressive feat of the tracking algorithm in this test is its ability to maintain good tracking characteristics when the egomotion of the camera changes direction at frames $550 - 650$. The difference between this situation and the situation at frames 200 and 1000 is that the change is smooth, rather than abrupt. Since we are estimating using an linear model, there is a big difference in the two scenarios. It should be noted that between frames 600 and 720, the perceived target location in the image plane is moving at 62.3 pixels per second, or alternatively 17.8% of the total image width per second. Being able to track the target with a small tracking error at such speeds, and not drift away from the target location when the perceived movement of the target suddenly changes direction, indicates that the choices made for R, Q and P were appropriate for this scenario.

The last and final test was the scenario where both the target *and* the camera is moving. During this test, the most important thing to evaluate was the tracking algorithm's ability to track the target when the camera was moving in one direction, and the target in another. To not mix up the tracking errors due to the small high frequent changes in actual location and the slow but large change in actual position, this video sequence was recorded without walking with the camera. Hence, it was only turned from side to side, and up and down. The target was moving from side to side at all times, in effect introducing the same problem as with the test where only the target was moving. That is, the classifier did not recognize the human in the cases where he was in the process of making a step. Hence, this scenario can be considered a test on how the tracking algorithm performs with a limited measurement rate, *and* a significant rate of change in the real object position. It should be noted that in this case, to achieve tracking at all, the normalization factor had to be changed in order for the BC/HL to classify the target correctly. This illustrates one of the weaknesses of the BC/HL classifier, and why the use of SVM/HOG is not always a bad idea. Another solution may be to implement some sort of adaptive normalization. However, this was out of the scope of the present thesis. Hence, the normalization factor was changed, and the performance of the tracking algorithm was logged. It is readily seen from figure 6.14 that the overall tracking can be considered successful also in this case. There is a loss of target tracking in frames 580-650 due to the fact that the target is standing so close to the border of the image, that it is occluded by not being entirely within the image. This is where the navigation algorithm would have been of great use. However, as mentioned, the navigation algorithm plays only a passive part in this case. The visual queue that the navigation algorithm gives in order to warn the control operator of that the orientation of the camera should be changed, can be seen in figure 6.10. This visual sign appears before the tracking of the object is lost,

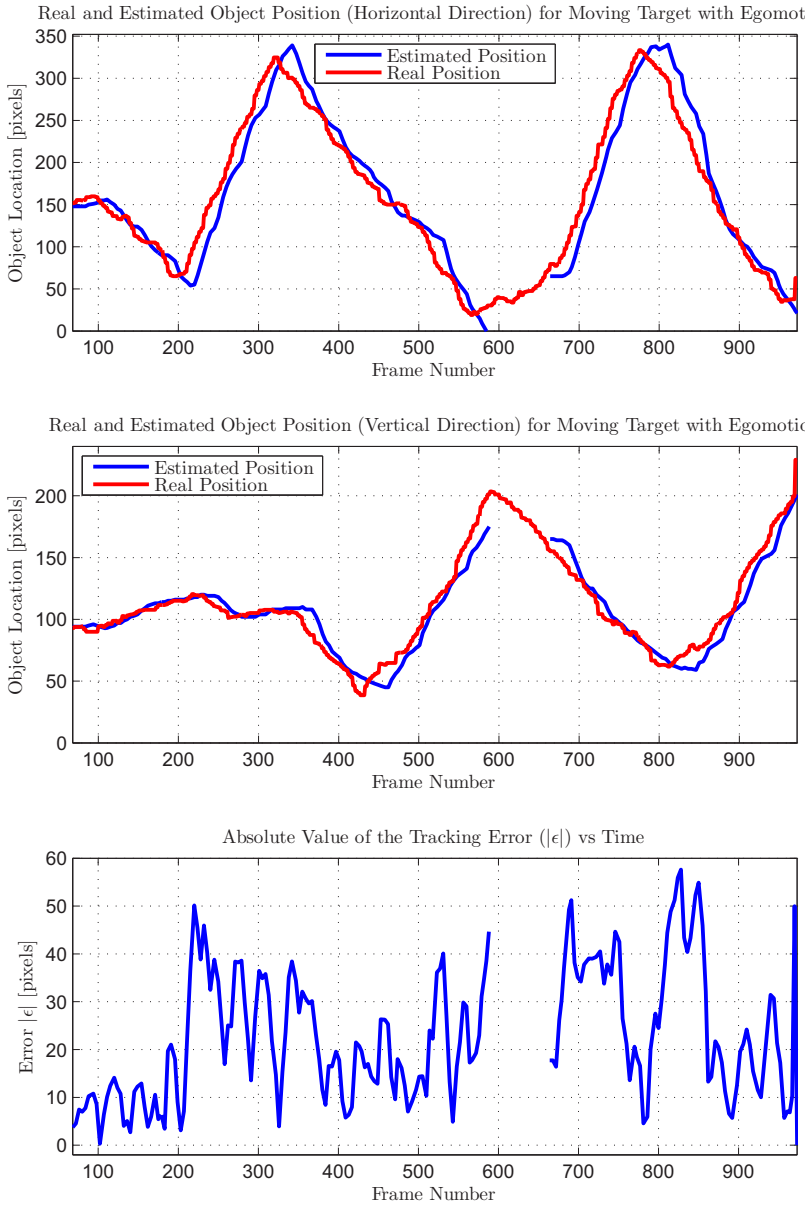

Figure 6.14: Simulated real-time test where both the camera and the target is moving. The camera movement is restricted to turning from side to side, or up and down. The target is moving from side to side. It is seen that with the exception of the case where the target is very close to the border of the image, the target tracking may be considered successful.

illustrating that if the navigation algorithm was able to send control signals to a camera position controller or alternatively the UAV autopilot, the loss of tracking would probably have been avoided.

The video streams that were sent to the command station during these tests, in addition to a real-time simulation on a desktop computer is attached to the electronic version of the present thesis.

This page intentionally left blank.

Chapter 7

Conclusion

The following chapter provides a brief overview of the process and main results of implementing the payload and the object detection and tracking algorithms. Subsequently, it reviews the most important findings from training and testing the two classifiers described in chapter 2. Finally, it concludes the thesis by stating the most important contributions and limitations of the work that was done.

7.1 Overview

In the present thesis, the main objective was the design and implementation of a small, light weight and power efficient payload system for UAVs. The payload should be able to perform real-time multiple object tracking based on thermal imaging while communicating with a ground station. The design and implementation of a ground station that was able to display a live video stream from the payload, along with important payload information such as CPU load and battery life was also of great interest.

Now, a payload was designed by initiating dialogs with firms specializing within the field of UAVs and robotics, as well as searching the Internet extensively for suitable components. Each component that was found to be suitable for use in

the payload, was subject to an evaluation process. This evaluation was based on criteria considered important for such components. The infrared camera module was especially emphasized in this process, as it was one of the most important components in the payload. After the components or component type for each module in the payload was decided, connection diagrams for both the data flow as well as power connections was specified. Finally, two different configuration setups for the payload was proposed, where each setup is intended for different scenarios. The first setup is intended for use in small UAVs where the allowed weight of the payload is small. The second setup configuration is for use in larger UAVs, where the allowed payload weight is in the range of $1kg$ or more. A software package for the payload was also proposed, where the focus was that the software should be reliable, easy to modify and up to date. This resulted in a generalized payload which can be easily modified, hence also be of use in a large variety of applications and different scenarios.

Having designed a payload suitable for the use in UAVs, the implementation and testing of a real-time tracking algorithm was important. The implemented object tracking algorithm is based on estimating the position of the tracked objects based on measurements of their actual position, resulting in an *estimate-and-measure* tracking approach. Since the payload, in spite of having limited computational power available, should be able to track objects in real-time, simple and efficient solutions were preferred. The simpler solutions may not track objects as efficient as state-of-the-art solutions within the field of computer vision, but advanced algorithms were not suited for implementation in the payload. This resulted in that the estimations were based on a *linear motion model*, and the estimation step of the tracking was performed by the implementation of a standard *Kalman filter*. Actions were taken to reduce the required computation power further, which consisted of manipulating the processed thermal images in such a way that the interesting objects could be localized without the need of performing object detection on the whole image. This was done by applying standard image processing functions such as *normalization*, *thresholding* and *connected components labeling*. Finally, the data association problem of the tracking algorithm (which measurements originates from which objects) was solved using a *global nearest neighbor* approach. This resulted in a fast tracking algorithm suitable for use in the payload.

To supply the tracking algorithm with measurements of the current location of the tracked objects, two different types of object detection and recognition algorithms were implemented. The first approach was based on a combination of the *histogram of oriented gradients* feature representations, and applying a *support vector machine* algorithm in a training process. The second object detection and

recognition algorithm consisted of a *boosted cascade* of classifiers, combined with the *Haar-like* feature representation. To test the effect of the training process for the two different approaches, a training set was acquired. This set consisted of 500 positive examples and 3000 negative examples, together with a test set which consisted of 150 images. The test images contained anywhere from 0 to 4 instances of the object that was to be recognized. Based on the training set, several classifiers were trained for each of the two object detection algorithms, and the performance of these classifiers was tested. Finally, a brief review of the results from this process was made by comparing the two classifier types to each other.

Having implemented the tracking algorithm and verified that the payload was working as intended, some extra features were also implemented. A *video streaming module*, whose job is to send the processed images (with included visualization of the Kalman filters and measurements) from the payload to the ground station was designed and implemented. The video stream is sent over the $HTTP$ protocol, and interfaced to the ground station through the ground station software $Neptus$. Furthermore, a *data logger module*, which logs the position trajectories of the tracked objects was created. The trajectories are stored locally on the payload's storage unit, but can be viewed in programs such as Matlab after a tracking test is performed. This is achieved by storing the position of the objects in the image plane at each processed frame. Each point is also matched to a frame number, effectively making it easy to compare the tracked position trajectory to the content of the video stream received at the command station. Finally, as a 'proof-of-concept', a naïve navigation algorithm was implemented, simply to test if the communication between the tracking algorithm and the navigation software *Dune* was working. This navigation algorithm should be made more advanced to achieve better control of the UAV, but the communication between the navigation algorithm and the auto pilot software will remain the same.

7.2 Evaluation of the Two Classifiers

From the evaluation of the performance of the classifiers described in chapter 6, some conclusions can be made. Specifically, given a test set of the same size as the one used in the present thesis, the BC/HL classifier will typically have both a smaller TPR *and* FPR compared to that of the SVM/HOG classifier. Now, since the tracking algorithm initiates tracking based on any measurement likely to originate from a target currently not being tracked, a large FPR yields a large

number of initiated Kalman filters. This, in turn, results in a large amount of regions that have to be searched and less computational power available to track actual objects. This is why it is very important that the FPR of the classifiers used in the detection step of the tracking algorithms is fairly low (e.g ≤ 0.5 false positives per frame). Of the classifiers trained, none of the SVM/HOG classifiers satisfy this criterion, hence they are *not suited* for use in the detection step of the tracking. For the BC/HL classifiers, it is seen that even though the TPR is not especially high, the FPR is very low. Since the tracking algorithm only requires one measurement every 5 frames to maintain tracking, this classifier may be used to achieve a reasonable tracking performance. Furthermore, from the results found during the classifier training, it is reasonable to conclude that the mirrored test set *should* be included in the training process. This can be seen from the ROC curves of the trained classifiers. However, it is also seen that adding *novel data* is a more efficient way to increase the classifier performance. Finally, it was found that training the classifiers having *all available data included* in the training set, yielded the best result for both classifiers. This is readily seen from the large decrease in FPR compared to the decrease in TPR. This means that the net change in performance is for the better. Furthermore, it should be noted that a larger training set would increase the performance of both classifiers, and that well trained classifiers of both types exist. Having evaluating the performance of the classifiers trained in the present thesis, it is reasonable to conclude that *more training data* should be added to the training set. This will increase their performance and create classifiers more suited for application in the detection step of the tracking algorithm. However, if restricted to a test set of similar size to that which was used in the present thesis, a BC/HL classifier should be the type of classifier that is trained.

A comparison of the two implemented detection algorithms irrespective of their TPR and FPR was also performed in the interest of future work. In the previous chapter, it became apparent that both algorithms have their weak and strong sides. However, the SVM/HOG classifier was proven to require too much computation to supply the tracking algorithm with a sufficiently high measurement rate in some of the simulated real-time scenarios. Specifically, while using the SVM/HOG classifier for the detection step, the real-time tracking algorithm could not keep track of the targets that were moving at a relatively high speed. This means that in the case where it is expected that there will be large variations in the actual object position, the BC/HL classifier is preferred for detection and tracking. Now, in the case where there are slowly moving targets that are tracked, the SVM/HOG should actually be used. The reason for this is that this classifier is more robust to changes in orientation and intensities. This makes the detection (not considering the detection speed) based on this classifier superior to that of the BC/HL classifier. To

extend this argument, the BC/HL classifier can only perform decent classification on objects very similar to the positive examples in the training set. This implies that it can only be used successfully in the tracking algorithm in scenarios where the appearance of the image *always* is the same. For many objects, this is not the case. Hence, this classifier becomes useless in some scenarios. To conclude this evaluation, it is clear that they are both useful, and the detection algorithm that should be applied is completely dependent on the test scenario. It should be noted that by minimizing interlacing effects in the thermal images, the performance of the SVM/HOG classifier will be increased. This can be done either by extra computation in the SBC, or alternatively internally in some frame grabbers.

7.3 Contributions

The main contributions of the present thesis is the design and implementation of a payload system, in addition to an efficient real-time tracking algorithm based on thermal imaging. The payload is heavily modularized, making it easy to change or modify one module without affecting others. This results in that the payload can be used for many different applications, and supplies the user with enough computational power to perform tasks such as real-time object tracking *online* from the UAV. The tests that were performed indicate that the total payload system was in many cases able to perform almost perfectly simulated real-time tracking of a human. However, it should be noted that during fast, abrupt and relatively large displacement rates in the actual position of the tracked targets, the tracking algorithm occasionally fail to keep track of the object. This is very reasonable, as the estimates are based on a *linear* motion model. The observed movement does not correspond to a linear motion, hence the abrupt change in velocity is not expected. Now, when the tracking is lost, reinitiation of the tracking often took some time due to the fact that tracking of new objects is initiated with a velocity estimation of 0. This resulted in a very inaccurate initial estimation of the object position, hence, the tracking algorithm ended up searching for the object at the wrong place. In the cases where the tracked target had more smooth movements, the real-time tracking performance was very good, having an average tracking error in the range of only ~ 30 pixels. This illustrates that even though the actual movement do not correspond to the motion model used for estimation (linear movement), good tracking can still be achieved. This is because of the fact that the Kalman filter, when its parameters are set appropriately, can be said to be very forgiving when it comes to inaccuracies in the motion model. It is clear that the only require-

ment of the motion model is that it is *accurate enough*. Furthermore, consistent tracking (i.e target is never lost) was achieved in almost all of the test cases where the change in actual location of the tracked target was smooth, failing only in the rare case of occlusions of the target (e.g when the real position is almost outside the image frame). Finally, the video streaming, data logger and the navigation algorithm were all working as intended, and the payload did not malfunction even once during testing. Considering this and the simulated real-time performance of the system as a whole, the design and implementation of the payload and its software can be concluded to be a success. However, there are definitely improvements that can be made. This is especially true for the process of making the tracking algorithm more robust with respect to changes in the average intensity value of the pixels in an image. Even though manually varying the normalization factor for each test scenario resulted in improved tracking performance, a more robust approach to this problem should be evaluated.

As a bi-product of the work performed in the present thesis, another contribution is that of a software bundle. This bundle contains software for many of the subtasks that had to be done during the work that is presented. These are tasks such as creating ROC curves for both the SVM/HOG and the BC/HL classifier types, and extracting positive and negative example images from a video sequence. Software to create the actual position trajectory of the tracked targets in a video sequence was also needed, along with a software able to extract the estimated position trajectories of the tracked target from the payload. Furthermore, this software proceeds to compare the two trajectories, and plots the tracking error in a graphical plot. All of this was done to make the process of comparing different parameter values in the tracking algorithm as autonomously as possible, and will probably be used to a great extent in future applications. The image capturing library used to read images from the raw video signal originating from the USB frame grabber is also something of practical use, even though it must be corrected for synchronization problems.

As a final note, it should be emphasized that the real-time tracking algorithm can be viewed as a mere framework for future development. Since it is based on an estimate-and-measure approach, it does not care whether the measurement originates from a SVM/HOG classifier, a BC/HL classifier, or any other technique able to obtain the measurement of the location of an object. In practice, this means that the tracking algorithm can be used to compare the performance of many different object detectors, novel as well as old. Furthermore, this framework can be used to track a large variety of different objects with a large variety of motions, simply by changing the object detection approach to something suitable for detecting

the object that is being tracked. The main limitation of this framework is that it bases its estimated target positions on a linear motion model. As shown in the presented test scenarios, this is in some situations an inaccurate motion model for the perceived target motions, which in turn inhibits the tracking performance.

This page intentionally left blank.

Chapter 8

Future Work

The payload, object detection and tracking algorithms presented in this thesis can be developed further in a number of ways. The payload provides flexibility with respect to the number of modules included in the setup, which means that new modules can be added without too much redesign. Furthermore, the object detection and tracking algorithms are subject to change, and can be modified or extended to increase the real-time tracking performance.

The following chapter features a review of the advantages of realizing setup configuration 2, and proceeds with suggestions on how the object detection algorithms can be extended. Finally, the chapter ends with a review of how the tracking algorithm can be improved, and how its functionality can be extended to be of more practical use.

8.1 Realization of Setup Configuration 2

The functionality of the payload can be extended by realization of the proposed setup configuration 2, and should be of a high priority in future work. One example of the extended functionality is the ability of the Axis M7001 image frame grabber to perform *deinterlacing* before the signal is broadcasted over the local network.

This will increase the performance of the classifiers, and by doing so also increasing the performance of the tracking algorithm. Furthermore, in chapter 4 it was mentioned that it is possible to develop a circuit board which provides the frame grabber with the correct voltages directly, avoiding the PoE interface. This should be kept in mind, as the step up converter from $5V$ to $\sim 40V$ was not tested, and may not be an optimal solution to the integration of the Axis M7001 encoder. Now, since the Axis frame grabber streams the capture video over the HTTP protocol, this video can be directly interfaced to the Neptus software located at the command station. This means that the control operator will have access to the video stream at two different HTTP addresses, where one stream is the processed images, and the other is the full video stream. This means that the stream coming directly from the axis encoder will be a $30fps$ stream, while the stream coming from the SBC in the payload will have as many fps as it has images processed per second. This results in a redundancy of the video stream which is not available in setup configuration 1. Now, in addition to this, the visible light video stream will also be made available at the ground station in the same manner as the infrared video. This is because the visual video camera can also be connected to an Axis M7001 encoder. Another advantage of introducing the M7001 frame grabber is that the SBC in the payload does not have to worry about processing the image stream coming from the cameras any more. This, in addition to that Dune is not running on the SBC in this setup, frees some computational power on the SBC. The extra computational power can be utilized by the object detection and tracking algorithms by introducing parallel computing. This allows for more complex detection and tracking algorithms, which in turn may yield a better tracking performance.

8.2 Extending the Object Detection Algorithms

With extra computational power available, the detection algorithms can become more advanced, and still be able to supply the tracking algorithm with a suitable rate of measurements. The first change of the object detection algorithm that should be researched, is the use of *contour extraction* rather than thresholding and connected-components labeling for the step of localizing promising regions in an image. The latter approach requires some fine tuning to work properly, and is therefore not very robust with respect to changes in the intensity values of the pixels. Now, a contour based approach is the principle of detecting the edges in an image, and localizing contours that are similar to those of the object that is to be detected. This requires more computational power than the thresholding

approach, but not more than what is available on the PandaBoard. This is especially true if parallel computing is used. In this case, there can be two parts of the object tracking algorithm running at the same time. This means that the process of finding the contours (edges) in an image, and localizing objects with similar size and shape to that of the object being tracked, can be performed in one CPU core. The process of tracking and estimating can then be done simultaneously by dedicating the other CPU core for this task. This will create a more continuous stream of measurements, and will probably increase the rate at which measurements are supplied to the tracking algorithm. To build further on the idea of parallel computing, the use of a graphical processing unit (GPU) for the detection step is a good idea. Implementing the object detection in the GPU allows for a great amount of parallelization, which would result in the ability to perform multi scaled object detection in the *same time* as the CPU use to perform *single scale* object detection. The PandaBoard has a GPU that is able to compute HOG descriptors and check the value of Haar-like features, but the developer of the SoC on the PandaBoard has not yet released the required drivers to the public. This means that if GPU programming is to be introduced, the SBC in the payload must be changed. However, in the third quarter of 2013, the Tegra4 platform is due to be launched. Tegra4 is an ARM platform with a CUDA GPU, effectively making it easy to implement GPU support in the payload. Hence, this is something that will be reviewed at a later time.

Another idea is to include multi-class classification. This can be introduced in various ways, but is intended for use in two different scenarios. One of the scenarios is the case where it is wanted to track different types of objects. The second scenario is the case when a BC/HL classifier is used for detection, but it is necessary to detect e.g cars oriented in different ways, i.e horizontally *and* vertically. To implement multi-class classification in the second case, separate BC/HL classifiers can be trained for each orientation, and then both of the classifiers can be used to classify the processed images. Each classifier will then return measurements for their respective orientation. In this way, in effect, an orientation invariant classifier is made.

Another interesting field is the field of sensor fusion. Sensor fusion in the case of the payload, refers to the concept of merging information from the thermal images with the information from the visual light video images. Since the IR camera is a LWFLIR camera, the IR camera and the video camera has complementary information. This results in that in some cases, the information can be combined to achieve a synergy effect. One such application is i.e the case where two humans are standing close to each other, continuously changing positions. If they are wearing

different colored clothes, the information from the video camera can easily be used to identify which human is which, while the tracking is based on the thermal images.

Finally, having established the weak and strong sides for each classifier type, an object detection algorithm that combines the SVM/HOG classifier with the BC/HL classifier could be of interest. The main problem with the application of the SVM/HOG classifier in the real-time tracking algorithm was the processing time during the detection step. Now, consider the case where the SVM/HOG classifier and the BC/HL classifier are alternating between being used for the detection step. This may result in a high enough rate of processed images, while still providing the tracking algorithm the benefit of being robust with respect to changes in intensities, and also to some degree object orientation.

8.3 Increasing Performance of the Object Tracking Algorithm

As discussed in chapter 6, the performance of the object tracking algorithm may be increased by using a more scientific method to determine the parameters of the Kalman filter. Several methods have been proposed to do this in the literature, but no universal best approach exists. Now, a method that may produce a promising result regarding the choice of Q, is to define this matrix according to equation 6.3. Having done this, the choice for σ_a can be posed as an optimization problem. I.e, by running a simulated real-time tracking test on a video sequence, and by registering the resulting tracking error, an optimization algorithm will be able to iterate towards the global optimum. However, it should be noted that doing such an optimization problem using only one video sequence will only find the optimal choice for Q for that specific sequence. To get a more generalized result, the optimization should be run over the course of $20 - 30$ different sequences all describing the same scenario. In this way, the bias from each video sequence may be reduced (or eliminated), and a better estimate of the global optimum is found. This approach can be extended to included the parameter choices for both P and R, which is also a case that should be studied.

The simulated real-time tracking tests performed in the present thesis revealed that there is a problem with that the initial predicted position and corresponding search region may not be very accurate. A way to solve this problem would be to introduce an adaptive scaling of the search region. An adaptive scaling of the

search region should function in such a way that the search region is chosen very large if the predicted state is uncertain. Similarly, if the certainty of the predicted location increases, the search region should decrease in size. The use of the matrix $S(k)$ as introduced in equation 3.14 should be considered as a candidate for this, but should probably be somewhat modified to correspond to the actual appeared object size.

As a general note, it should be a priority to make the tracking algorithm more autonomous with respect to the choices of the parameters such as the normalization factor and the number of frames between each full image search, as these are crucial parameters achieving successful tracking. For example, if the tracking algorithm currently is not tracking any objects, there are no reason for the tracking algorithm to completely skip the intermediate frame between each fifth frame. On the contrary, performing a full image search in every frame in such cases will probably increase the tracking performance, as (re)initiation of the Kalman filter will occur faster. Furthermore, in situations where the predicted estimates are very uncertain, a solution would be to increase the rate at which the full image is searched, in effect removing the problem of having a badly predicted estimate of the object position.

8.4 Extending the Object Tracking Algorithm

An extension of the object tracking algorithm that should be implemented and tested is the implementation of a *particle filter* based estimator. This is an estimator type which has proven useful in the estimation of non-Gaussian and non-linear processes. The particle filter has become a popular way of tracking objects in computer vision in recent years, but it also requires a lot more computational power than the linear Kalman filter. However, as previously mentioned, the SBC has more free computational power when the payload is realized with setup configuration 2. Splitting the tracking algorithm into two separate threads, such that each thread can be run in separate CPU cores, it should be tested if the PandaBoard has sufficient computational power for estimation when using the particle filter in an efficient way. The effect of introducing the particle filter on the measurement rate should also be evaluated.

Now, the most interesting extension of the object tracking algorithm, would be to extend the predicted location from two dimensional to the three dimensional

space. That is, instead of estimating the target position in the image frame, the real world coordinates of the targets are estimated. It should be noted that to still achieve good tracking performance in the image plane, it may be necessary to keep two separate estimates. That is, having one estimator for the position in the image plane, and then another estimator for the real world coordinate. The first thing required for this to function properly is a GPS/INS device. As it already is a part of the suggested payload design in chapter 4, this process is reduced to finding a suitable component. After this is done, some synchronization algorithm has to be implemented, effectively synchronizing the GPS/INS measurements to specific processed image frames. Once this is done, the estimation of the real world coordinates of the tracked objects can commence. Using a simple approach, it can be assumed that the tracked target is positioned at the ground. Furthermore, the ground can be estimated as a flat plane, effectively leaving all the tracked objects in the same attitude coordinate. Now, having an estimate for the image position of the object, together with an estimate of the attitude of the UAV, the process of estimating the real world coordinates of the human is a simple problem of triangulation. After having calculated the real world coordinates of a tracked object with this approach, these coordinates can be passed on from a measurement to an estimator that is responsible for estimating the real world coordinates of the target. It should be noted that this approach relies on accurate measurement of the attitude. Hence, methods to acquire the most precise attitude measurements should be looked into. One of the key advantages of estimating the real world coordinates of the tracked objects, is that if the tracking of the object is lost, the estimate of the last known real world position is still known. This makes the process of regaining tracking of the object easier, as the UAV now is more aware of where it should search for the missing target.

Appendices

Appendix A

Discrete Convolution

In the discrete case, convolution is a mathematical operation on two functions f and g, which produce a third function that can be viewed as the sum over all space of the function g at m times the function f at $n - m$. That is, the result of convolution is a function $c[n]$ which can be defined as

$$c[n] = (f \star g)[n] \coloneqq \sum_{m=-\infty}^{\infty} f[n-m]g[m] \tag{A.1}$$

And when g has finite support in the set $\{-M, -M+1, ..., M-1, M\}$ a finite summation may be used:

$$c[n] = (f \star g)[n] \coloneqq \sum_{m=-M}^{M} f[n-m]g[m] \tag{A.2}$$

Appendix B

Focal Length and Perceived Object Size

The focal length f of a lens, is the distance between the center of the lens element to the image sensor surface. This distance is illustrated in figure B.1. Now, the relationship between the focal length f, the object size A, the distance d between the lens and the object, and the resulting projected object size s_o on the image

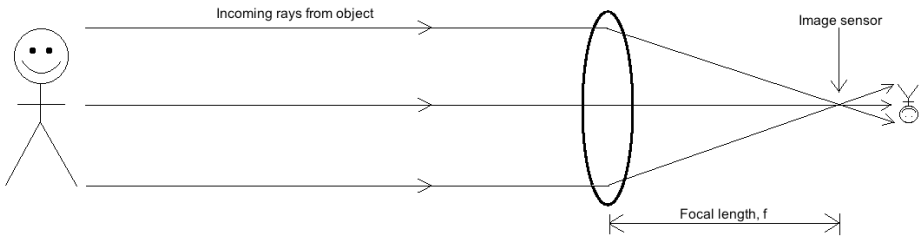

Figure B.1: An illustration of the relative position between the focal length and the image sensor.

sensor is

$$\frac{f \times A}{d} = s_o \tag{B.1}$$

And to find the perceived object size in terms of pixels in the image, the following relation holds true

$$\frac{s_o}{s_{tot}} \times p_{tot} = p_o \tag{B.2}$$

where s_o is as previously defined, s_{tot} is the total size of the image sensor, and p_o is the object size in image pixels. Combining equations B.1 and B.2 we get

$$p_o = p_{tot} \times \frac{f \times A}{d \times s_{tot}} \tag{B.3}$$

Notice that doubling the focal length f has the same effect in terms of perceived object size, as halving the distance between the lens and the object.

Now, consider the case where an UAV is flying at an altitude of $50m$, with an infrared camera mounted on the side of the plane, pointing 45° downwards. Assume further that the infrared camera is the Tau2 640, described in appendix D, with a $13mm$ focal length lens. Now, assuming that a human of height $1.8m = 1800mm$ and width $0.5m = 500mm$ is present, and that the human is projected exactly at the center of the lens. Then, the distance d from that human to the camera lens is equal to

$$d = \frac{50m}{cos(45°)} \sim 70m = 70000mm$$

Now, from appendix D, it is seen that the size of the image sensor in the vertical direction is equal to $17\mu \times 512 = 8.7mm$. The total image height in pixels p_{tot} is equal to 512. Putting everything together, it is readily seen that the perceived object size in the terms of pixels in the vertical direction is equal to

$$\frac{13mm \times 1800mm}{70000mm \times 8.7mm} \times 512\, pixels \sim 20\, pixels$$

Performing the same calculations for the horizontal direction yields a value of 11 pixels, effectively making the perceived object size $11x20$ pixels. By the same logic, the same human would be $8x16$ pixel if the infrared camera was changed to Tau2 336 with a $9mm$ focal length lens.

Appendix C

Single Board Computer

The single-board computer used in the present thesis is the PandaBoard. The PandaBoard is based on the Texas Instruments OMAP4430 SoC, and is a mobile software development platform. Following is some of its most important properties.

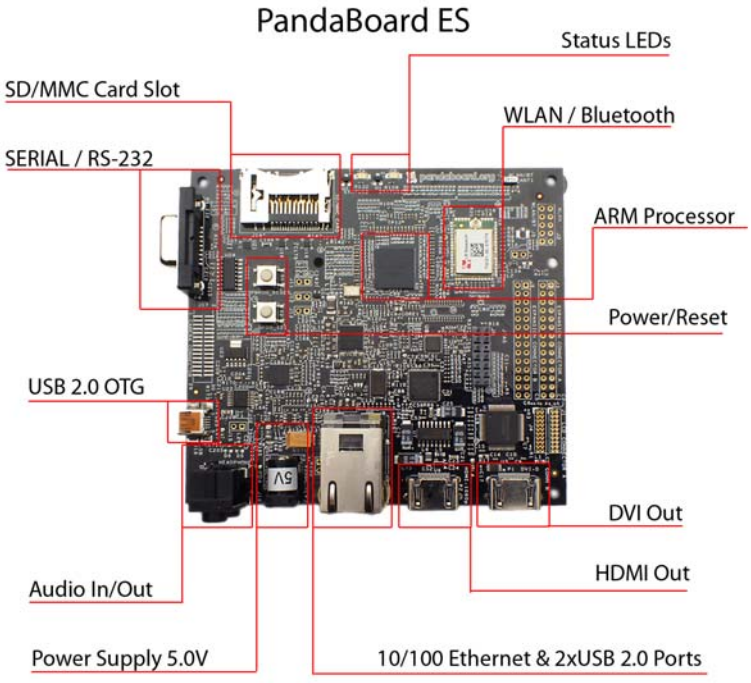

Figure C.1: The PandaBoard and its most important specifications.

Overview	**PandaBoard ES**
CPU	1.2GHz Dual Core
Processor Type	ARM Cortext-A9
RAM	1GB - GPU
GPU	304 MHz PowerVR SGX540
Physical Attributes	
Size	$11.4cm \times 10.1cm \times 3cm$
Weight	82g
Interfacing	
Display Port	HDMI
Power	DC Jack / USB OTG - 5V ~ 1A
Storage	SD Card - Ethernet WiFi
Ethernet	RJ45 and WiFi
Environment	
Operating Temperature	$-20° - 70°$
Humidity	N/A

Appendix D

Thermal Imaging Camera

Figure D.1: The FLIR Tau2 Thermal Imager

Overview	**FLIR Tau 640**
Analog Video Display Formats	640 x 480 (NTSC); 640 x 512 (PAL)
Pixel Pitch	$17\mu m$
Spectral Band	$7.5 - 13.5\mu m$ (LWIR)
Frame Rates	7.5 Hz NTSC; 8.3 Hz PAL
Sensitivity	≤ 50 mK at f/1.0
Scene Range	-25°C to +100°C
Physical Attributes	
Size (w/o lens)	1.75" x 1.75" x 1.18"
Weight	72g
Interfacing	
Input Power	4.0 - 6.0 VDC
Power Dissipation, steady state	~ 1.2 W
Environmental	
Operating Temperature Range	−40°C to +80°C
Temperature Shock (5°/min)	Yes
Operational Altitude	Up to 12km
Humidity	Non-condensing between 5% - 95%
Vibration	4.3g three axis, 8 hr each
Shock	200g shock pulse w/ 11 msec sawtooth

Overview	**FLIR Tau 336**
Analog Video Display Formats	640 x 480 (NTSC); 640 x 512 (PAL) (Up-sampled from 336×256)
Pixel Pitch	$17\mu m$
Spectral Band	$7.5 - 13.5\mu m$ (LWIR)
Frame Rates	7.5 Hz NTSC; 8.3 Hz PAL
Sensitivity	$\leq$ 50 mK at f/1.0
Scene Range	-25°C to +100°C
Physical Attributes	
Size (w/o lens)	1.75" x 1.75" x 1.18"
Weight	72g
Interfacing	
Input Power	4.0 - 6.0 VDC
Power Dissipation, steady state	$\leq$ 1.0 W
Environmental	
Operating Temperature Range	−40°C to +80°C
Temperature Shock (5°/min)	Yes
Operational Altitude	Up to 12km
Humidity	Non-condensing between 5% - 95%
Vibration	4.3g three axis, 8 hr each
Shock	200g shock pulse w/ 11 msec sawtooth

Appendix E

Video Camera

Figure E.1: The GoPro Hero2 Video Camera

Overview	**GoPro Hero2**
Analog Video Display Format	848x480 pixels
Frame Rate	30
Field of View	$127° \times 90°$
Physical Attributes	
Dimensions	$4.2cm \times 6.0cm \times 3.0cm$
Weight	96g
Interfacing	
Power	Internal Battery
Output 1	HDMI and Analog
Output 2	Combo stereo audio and composite video

Appendix F

Frame Grabbers

Figure F.1: Two different frame grabbers, with two different interfaces.

Overview	**Axis M7001**
Video Compression	H.264 or MJPEG
Resolutions	720×576 to 76×144
Frame Rate	30/25 (NTSC/PAL)
Video Streaming	Both H.264 and MJPEG
Supported Protocols	IPv4/v6, HTTP, HTTPS, FTP, SMTP, UPnP, TCP, UDP, RTCP, DHCP ++
Physical Attributes	
Weight	82g
Dimensions	$10,1 \times 8.6 \times 3.7 cm^3$
Interfacing	
Power	Power over Ethernet (Class $2 \sim 5W$)
Connectors (Input)	Analog composite video (BNC input)
Connectors (Output)	Ethernet (RJ45)
Envoirmental	
Operating Conditions	$0° - 50°$ C
Humidity	20-80%

Overview	**EasyCap DC60**
Video Compression	AVI or MPEG
Resolutions	720×576 to 76×144
Frame Rate	30/25 (NTSC/PAL)
Video Streaming	Both AVI and MPEG
Supported Protocols	USB
Physical Attributes	
Weight	57g
Dimensions	$8.8 \times 2.8 \times 1.8 cm^3$
Interfacing	
Power	2.5W
Connectors Input	Analog composite video (RCA)
Envoirmental	
Operating Conditions	N/A
Humidity	N/A

Appendix G

Battery

Figure G.1: Biltema PowerPack, with and without the chassis

Overview	**Biltema PowerPack**
Battery Type	Litium-Polymer
Capacity	2000mAh
Maximum Current Output	1A
Physical Attributes	
Size	$9,3cm \times 5,0cm \times 1,7cm$
Weight	75g
Weight w/o chassis	45g
Interface	
Input	5V Micro-USB
Output	5V USB
Envoirmental	
Temperature	0° – 50° C
Humidity	Max. 90%

Appendix H

Ethernet Switch

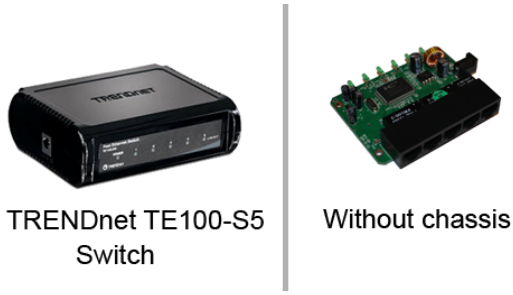

Figure H.1: The TRENDnet TE100-S5 Switch, with and without its chassis

Overview	**TRENDnet TE100-S5 5 Port Switch**
Standard	IEEE 802.e 10Base-T IEEE 802.3u 100Base-TX IEEE 802.3x Flow Control
Network Media	Ethernet Fast Ethernet
Data Rates	Ethernet: 10Mbps/20Mbps Fast Ethernet: 100Mbps/200Mbps
Physical Attributes	
Weight	128g (59g with no chassis)
Dimensions	$10cm \times 7.8cm \times 3.1cm$
Interfacing	
Input/Output	5 Ethernet Ports (RJ-45)
Power	2.8W
Envoirmental	
Temperature	$0^\circ - 50^\circ$ C
Humidity	Max. 90%

Bibliography

[1] Imc protocol documentation. https://www.lsts.pt/imc/doc/master/. Accessed: 15/06-2013.

[2] Minimum assignment problem solver. https://code.google.com/p/hungarian-cpp/. Accessed: 15/06-2013.

[3] Mjpeg streamer plugin. http://mjpg-streamer.sourceforge.net/. Accessed: 15/06-2013.

[4] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25 in Automation and Remote Control,, pages 821–837, 1964.

[5] Ahmed Ali and Kenji Terada. A general framework for multi-human tracking using kalman filter and fast mean shift algorithms. 16(6):921–937, mar 2010.

[6] Saad Ali and Mubarak Shah. Cocoa - tracking in aerial imagery. In *Proc. Int. Conf. on Computer Vision*, 2005.

[7] Saad Ali and Mubarak Shah. Cocoa - tracking in aerial imagery. In *Proc. Int. Conf. on Computer Vision*, 2005.

[8] Plamen Angelov, Chirag Gude, Pouria Sadeghi-Tehran, and Tsvetan Ivanov. Artot: Autonomous real-time object detection and tracking by a moving camera. In *Intelligent Systems (IS), 2012 6th IEEE International Conference*, pages 446–452. IEEE, 2012.

[9] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual*

ACM Workshop on Computational Learning Theory, pages 144–152. ACM Press, 1992.

[10] Christopher A. Brooks and Karl Iagnemma. Visual detection of novel terrain via two-class classification. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1145–1150, New York, NY, USA, 2009. ACM.

[11] José Carlos Castillo, Juan Serrano-Cuerda, Antonio Fernández-Caballero, and María T. López. Segmenting humans from mobile thermal infrared imagery. In *Proceedings of the 3rd International Work-Conference on The Interplay Between Natural and Artificial Computation: Part II: Bioinspired Applications in Artificial and Natural Computation*, IWINAC '09, pages 334–343, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] A. Cesetti, E. Frontoni, A. Mancini, P. Zingaretti, and S. Longhi. A vision-based guidance system for uav navigation and safe landing using natural landmarks. *J. Intell. Robotics Syst.*, 57(1-4):233–257, January 2010.

[13] Chi-Tsong Chen. *Linear System Theory and Design*. Oxford University Press, Inc., New York, NY, USA, 3rd edition, 1998.

[14] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *In CVPR*, pages 886–893, 2005.

[15] Weidong Ding, Jinling Wang, Songlai Han, Ali Almagbile, Matthew A. Garratt, Andrew Lambert, and Jack Jianguo Wang. Adding optical flow into the gps/ins integration for uav navigation, 2009.

[16] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, 2009.

[17] Xi Bin Wang et al. Obstacles avoidance for uav slam based on improved artificial potential field. *Applied Mechanics and Materials*, 241-244:1118–1121, 2012.

[18] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[19] A. Gaszczak, T. P. Breckon, and J. Han. Real-time people and vehicle detection from UAV imagery. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7878 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, January 2011.

[20] A. Gaszczak, T.P. Breckon, and J.W. Han. Real-time people and vehicle detection from UAV imagery. In *Proc. SPIE Conference Intelligent Robots and Computer Vision XXVIII: Algorithms and Techniques*, volume 7878, 2011.

[21] Modesto Castrillon-Santana Hannes Kruppa and Bernt Schiele. Fast and robust face finding via local context.

[22] T. Joachims. Making large-scale SVM learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, MA, USA, 1999.

[23] Thorsten Joachims. Advances in kernel methods. chapter Making large-scale support vector machine learning practical, pages 169–184. MIT Press, Cambridge, MA, USA, 1999.

[24] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[25] Pavlina Konstantinova, Alexander Udvarev, and Tzvetan Semerdjiev. A study of a target tracking algorithm using global nearest neighbor approach. In *Proceedings of the 4th international conference conference on Computer systems and technologies: e-Learning*, CompSysTech '03, pages 290–295, New York, NY, USA, 2003. ACM.

[26] Jing Li and Nigel M. Allinson. A comprehensive review of current local features for computer vision. *Neurocomput.*, 71(10-12):1771–1787, June 2008.

[27] Rainer Lienhart, Er Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *In DAGM 25th Pattern Recognition Symposium*, pages 297–304, 2003.

[28] Yuping Lin, Qian Yu, and Gérard Medioni. Efficient detection and tracking of moving objects in geo-coordinates. *Mach. Vision Appl.*, 22(3):505–520, May 2011.

[29] Gellért Máttyus, Csaba Benedek, and Tamás Szirányi. Multi target tracking on aerial videos. In *ISPRS Istanbul workshop 2010 on modeling of optical airborne and space borne sensors, WG I/4.*, pages 1–7, Istanbul, 2010. IAPRS.

[30] Ram Nevatia. Unsupervised incremental learning for improved object detection in a video. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3298–3305, Washington, DC, USA, 2012. IEEE Computer Society.

[31] J. Fernandez P. Wilson. Facial feature detection using haar classifiers. *JCSC 21*, 4:127–133, April 2006.

[32] Andrew Price, Jacob Pyke, David Ashiri, and Terry Cornall. Real time object detection for an unmanned aerial vehicle using an fpga based vision system. In *ICRA*, pages 2854–2859. IEEE.

[33] Ashraf Qadir, Jeremiah Neubert, and William Semke. On-board visual tracking with unmanned aircraft system (uas). *CoRR*, abs/1203.2386, 2012.

[34] Mohammad J. Saberian and Nuno Vasconcelos. Boosting classifier cascades. In *In NIPS*, 2010.

[35] D. A. Sadlier and N. E. O'Connor. Evaluation of a vehicle tracking system for multi-modal uav-captured video data. pages 76680X–76680X–12, 2010.

[36] Taek Lyul Song, Dong Gwan Lee, and Jonha Ryu. A probabilistic nearest neighbor filter algorithm for tracking in a clutter environment. *Signal Process.*, 85(10):2044–2053, October 2005.

[37] Rainer Stiefelhagen, Rachel Bowers, and Jonathan G. Fiscus, editors. *Multimodal Technologies for Perception of Humans, International Evaluation Workshops CLEAR 2007 and RT 2007, Baltimore, MD, USA, May 8-11, 2007, Revised Selected Papers*, volume 4625 of *Lecture Notes in Computer Science*. Springer, 2008.

[38] F. Suard, A. Rakotomamonjy, and A. Bensrhair. Pedestrian detection using infrared images and histograms of oriented gradients. In *in IEEE Conference on Intelligent Vehicles*, pages 206–212, 2006.

[39] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[40] Peter van Blyenburgh. Uavs: an overview. *Air & Space Europe*, 1(56):43 – 47, 1999.

[41] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[42] Antti Vyrynen and Justin Salminen. Lithium ion battery production. *The Journal of Chemical Thermodynamics*, 46(0):80 – 85, 2012. ¡ce:title¿Thermodynamics of Sustainable Processes¡/ce:title¿.

[43] Weihong Wang, Jian Zhang, and Chunhua Shen. Improved human detection and classification in thermal images. In *ICIP*, pages 2313–2316, 2010.

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

[45] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4), December 2006.

[46] Yudong Zhang and Lenan Wu. Classification of fruits using computer vision and a multiclass support vector machine. *Sensors*, 12(9):12489–12505, 2012.

[47] Haibin Zhu, MengChu Zhou, and Rob Alkins. Group role assignment via a kuhn-munkres algorithm-based solution. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 42(3):739–750, 2012.