Auto-Pilot: Autonomous Control of a remote controlled helicopter

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

S. A. Currie

Grahamstown, South Africa
November 2012

Abstract

An auto-pilot system was created which could track and control a Syma S107 RC mini infrared helicopter in real-time. The helicopter was tracked using two LEDs attached to it, one situated on the front of the helicopter and one situated on the tail. Image processing and object tracking algorithms were used to track the position of the helicopter. In order to perform real-time tracking, the methods used had to be very efficient. Thresholding was used for segmentation, so that the co-ordinates of the two LEDs on the helicopter could be extracted. CamShift was used to track the helicopter and use its position to limit the search space for the LEDs. The orientation of the helicopter was found using a combination of the distance between the two LEDs and the distance of the helicopter from the camera. The Kinect's depth sensor was used to determine the distance of the helicopter. The helicopter was controlled by reverse engineering its infrared protocol. An infrared LED was attached to an Arduino Uno board in order to send the infrared signals and control the helicopter from the computer. A closed-loop feedback system was created where the helicopter's position and orientation was used to make it perform certain movements, including hovering and flying in a horizontal square.

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System (1998 version, valid through 2012)

I.2.9 *[Artificial Intelligence]*: Robotics—autonomous vehicles
I.4.8 *[Image Processing and Computer Vision]*: Scene Analysis—tracking, colour

General Terms: Algorithms, Performance

Acknowledgements

I would like to thank my supervisor, Mr. James Connan for his help and support with this project. His extensive knowledge in image processing and his passion for the subject gave me the motivation necessary to complete this thesis.

I would also like to acknowledge Rhodes University for providing me with the necessary hardware and for the financial support received from the Dr. Kendall Scholarship and the Rhodes University Honours Scholarship. I would also like to acknowledge the financial and technical support of Telkom, Tellabs, Stortech, Genband, Easttel, Bright Ideas 39 and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

On a more personal note, I would like to thank my parents, Mark and Jean Currie. I am extremely grateful for their support and interest in this project. I would also like to thank Hayden Mackenzie for his constant encouragement and support throughout this year.

Contents

List of Figures

List of Tables

Glossary

API	Application Programming Interface
CMOS	Complimentary metal-oxide-semiconductor. It is used for creating integrated circuits
DARPA	Defence Advanced Research Projects Agency
GPS	Global Positioning System providing navigation information using satellites
HSV	A colour space made up of Hue, Saturation and Value
IDE	Integrated Development Environment
ISO	A digital sensor's sensitivity to light, usually in a digital camera
LED	Light Emitting Diode
PWM	Pulse Width Modulation
RGB	A common colour space which represents every colour as a certain amount of red, blue and green
USB	Universal Serial Bus
YUV	A colour space which has a luminance channel, Y, and two colour channels without brightness information, UV.

Chapter 1

Introduction

The importance of computer vision has been revealed through the many real life applications it makes possible. These range from medical applications, such as cancer detection, to security applications, such as biometrics. Humans use vision to perceive and understand the world and make decisions. Computer vision aims to copy this human sense, in order to process and extract information from images, which help it to understand and interpret the image. Computer vision can, in some cases, outperform human vision, due to the power of modern computers. Computer vision uses image processing and many other techniques to achieve this goal.

Object tracking is a subset of computer vision. It is the process of recognizing an object and following the movement of that object. There are many different methods and algorithms that can be used for object tracking. Some algorithms require particular conditions in order to work, but there are a number of algorithms which can be applied in many different contexts. This represents the great advantage of computer vision, where the skills learnt for one application can often be reused for many other, unrelated applications. The ability to track an object has opened up numerous opportunities for applications such as eye tracking, traffic flow management and autonomous vehicles. There has been significant interest in autonomous air vehicles (or drones), as they can be used for military purposes such as battlefield environments and for security and surveillance[17]. This research project uses autonomy as an application of object tracking.

1.1 Problem Statement and Research Goals

The aim of this research project is to create a system that simulates an autopilot system for a mini remote controlled helicopter. This helicopter must be tracked using a camera and sent messages in real time, in order to control it and make it fly autonomously. It will therefore be necessary to find a suitable method of object tracking. The main constraint of the chosen object tracking algorithms is that they have to be efficient, in order to track and control the helicopter in real-time. They also need to accurately determine the 3D position and basic orientation of the helicopter at any given time. The end result of the project will be a system that controls the helicopter, making it perform certain pre-defined movements.

1.1.1 Objectives

The main objectives of this research project are stated as follows:

- Send commands to the helicopter from a transmitter which will control its movements.
- Identify and track the helicopter to determine its 3D position and orientation.
- Use this tracking information to control the helicopter autonomously.

1.2 Thesis Organisation

Chapter 2 introduces some background to the relevant areas addressed by the project and reviews the previous work done in these areas. This section aims to give some insight into the different methods and algorithms which may be used for this project.

The tools and design used for the project are explained in Chapter 3. An outline is given of the software requirements, such as libraries and algorithms, and the hardware requirements necessary to perform the project.

The implementation of the project is explained in Chapter 4. This section gives a technical explanation of the methods used to carry out the project.

The testing of the system and the results are given in Chapter 5. Each test is described and its outcome is given. The results are analysed and the capabilities and limitations of the system are discussed.

Finally, a conclusion is made in Chapter 6 which summarises the results and outcomes of the system. Future work is also discussed.

Chapter 2

Background

Since this project is a first step into the world of automation, it is necessary to give a brief background of the different classes of autonomous vehicles and examples that already exist in these classes. Image processing and object tracking form the main focus of this project; therefore it is necessary to discuss the relevant literature and algorithms already existing in this area.

2.1 Autonomous Vehicles

Helicopters are fairly difficult to control due to their complexity [8, p. 1]. They are, however, very useful vehicles and have many real-world applications. Autonomous helicopters have many advantages and uses. They could be used for security, such as helping pilots in situations where they lose control of the vehicle, or provide surveillance of aerial space. They can also be used for monitoring environmental factors, such as volcanoes or air pollution. Another very pertinent use is to help in dangerous situations, such as removing mines and going into radioactive atmospheres [8, p. 1]. There has even been development in ocean observing systems[23], using autonomous underwater and air vehicles to monitor the ocean for oil spills, tsunamis and wreckage. Autonomous vehicles generally fall into three categories, namely on-board intelligence, external intelligence and combined on-board and external intelligence.

2.1.1 On-Board Intelligence

On-board intelligence is when all the tracking and navigation is performed within the vehicle. The Stanford entry in the DARPA Grand Challenge is an example of on-board intelligence. The Grand Challenge is a competition in which teams and organisations have to create an autonomous ground vehicle which can navigate its way around an off-road course within a certain amount of time. The vehicle, named Stanley, won the challenge. The rules of the challenge included that no manual intervention was allowed and the vehicles had to drive themselves [39]. The car had a GPS system, camera, antennae and sensors attached to its roof rack[39]. All the intelligence was implemented on board. Another example of on-board intelligence is an autonomous model helicopter created by [4] as part of a Ph.D thesis. This helicopter had on-board processors, video cameras, gyroscopes and accelerometers. All image processing and navigation was done on the helicopter itself. The disadvantage of this type of functionality is that on-board equipment can be quite heavy, so there may be weight concerns.

2.1.2 Remote or External Intelligence

External intelligence is when the vehicle is tracked by and sent navigation data from an external source. This type of intelligence requires a form of communication between the vehicle and the external source. The vehicle used may have weight limitations or size constraints, making on-board sensors unsuitable [32]. In this case, external devices need to be used. An optical 3D capturing system was created by [32] which was used to keep track of a mini remote-controlled helicopter. Many cameras were used in order to capture all angles of the helicopter. All the measurements and navigation decisions were done on an external computer, and information was sent to the helicopter from the computer using a serial connection. The disadvantage of this type of functionality is that if the vehicle is occluded by another object at any point, tracking can become difficult.

2.1.3 Combined External and On-Board Intelligence

External and on-board intelligence combines the above categories, meaning certain parts of the functionality may be performed externally and certain parts may be on board. For example, the tracking may be performed by an external camera and the navigation performed on board, or vice versa. A system was created by [3] in order to test a pose estimation algorithm for an unmanned four rotor helicopter. This system falls into this category, as two cameras were used to estimate the position of a remote controlled quad rotor helicopter; one of the cameras was located on the ground and the other was located on the helicopter. Each of these cameras was connected to different computers, which were connected by a network. The on-board camera sent information to the external computer through the network and the external camera sent information to the helicopter using a remote control device with a parallel port. The disadvantage of this is the added weight to the helicopter and a large amount of equipment is needed which may become expensive.

2.2 Image Processing

Image processing is the method of taking in an image as input and performing a number of operations on it in order to enhance it, or extract necessary information from it[1]. Digital image processing is when these actions are performed by a computer, instead of a human. The digital image which is used by the computer is made up of many elements called pixels. Analogue image processing is when these actions are performed by humans, but this is limited because human vision only includes the visual band of the electromagnetic spectrum. Computers, however, can process images which range over almost the entire electromagnetic spectrum (such as ultrasound, radio waves and electron microscopy) and are therefore much less limited [18].

2.2.1 Image Preprocessing

Preprocessing is performed on an image to try and correct or enhance certain features. There are four main types of preprocessing[35, pp. 108–111]:

- Brightness transformations - adjust the brightness of the image.
- Geometric transformations - remove distortions in the image. These distortions may be caused by the sensor being in the wrong position [35, p. 65].
- Local neighbourhood preprocessing - consists of smoothing to reduce unwanted noise and edge detection to find region boundaries.

[1]Engineers Garage, http://www.engineersgarage.com/articles/image-processing-tutorial-applications, 2012

- Image restoration - reduce image degradation. In the study by [38], an image restoration method was used to reduce the effects of fog, smoke or haze which cause an image to have low contrast and faded colours.

2.3 Object Tracking

Object tracking is defined by [42], as the estimation of a moving object's trajectory in an image plane. According to [42], object tracking methods can be divided into four main areas. These are object shape representation, feature selection, object detection and object tracking. Each of these areas can be implemented in a number of different ways and the methods chosen for each will differ according to the environment and context in which the tracking is performed. Before these areas are discussed, the main challenges of object tracking are first introduced to give an idea of what needs to be dealt with by the methods. Orientation estimation is also discussed, as this is often a very important part of object tracking.

2.3.1 Challenges in Object Tracking

The following list summarizes some of the main challenges that may be faced when tracking an object:

- Illumination changes - This is when the light or illumination changes throughout an image sequence. Change in illumination is a problem in object tracking as it affects the colour of the background and objects being tracked. This would happen frequently in an outdoor environment, but could be controlled in an indoor environment with constant lighting.
- Occlusion - This is when an object is hidden from view by itself, another object or the background [42]. Occlusion is a problem in tracking as the object may be lost at a certain time in an image sequence and the tracking algorithm needs to deal with this loss and detect the object when it comes back into view.
- Background motion - This is when the object being tracked is not the only moving object in the scene. Certain objects may also be moving in the background. In an outdoor environment, this could include moving branches, water, people or cars.

This presents a challenge in object tracking, as movement in the background could be mistaken for movement by the object being tracked. This could be controlled in an indoor environment, where the background is likely to be static.

- Noise - A “noisy” image is one with many unwanted speckles, giving it a grainy effect. This may be caused by the camera and can be affected by exposure and temperature[2]. Exposure is made up of the amount of light allowed through the lens (aperture), the length of exposure to the light (shutter speed) and the degree of sensitivity to the light (ISO). The temperature of the sensor may cause noise if it gets too hot. Noise is a problem for object tracking as it reduces the quality of the image and therefore reduces accuracy of the tracking. There will always be some noise in a captured image but it can be reduced greatly with good lighting. Images captured at night are likely to have a significant amount of noise. Noise can also be reduced in a captured image by using noise-reducing filters.

2.3.2 Object Shape Representation

There are many different ways of representing an object in an image. One representation is to use points, which are suitable for smaller objects. An interest point indicates a significant change in an image[16]. Interest points based on colour were used by [16] to represent objects, including soccer players and cars. Another representation makes use of primitive geometric shapes, which are generally used for rigid objects but may also be used to represent non-rigid objects. A system was developed by [10] to test a framework for the tracking of non-rigid objects. This system used a circular region to represent objects, including a ping-pong ball, a person’s upper body and a person’s face. A block matching object tracking algorithm was proposed by [37] which used a rectangle to represent a person. An object can also be represented by its silhouette or contour. This is suitable for non-rigid objects. The objects in [29], which included a person’s upper body and face, were represented by a contour which was made up of the edges of the object. Skeletal models can also be used to represent an object. These are suitable for articulated, rigid objects. An object can also be represented by a blob or many blobs with similar colour or flow [24].

[2]HubPages, http://carpesomediem.hubpages.com/hub/understanding-image-noise, 2011

2.3.3 Feature Selection

Certain features need to be extracted from an object in order to distinguish it from the rest of the scene. These features should be chosen according to the type and appearance of the object. Different aspects of the scene will also affect which features should be chosen, as large changes in illumination and noise can negatively affect features like colour.

Colour

Colour is the most popular visual feature that can be used. This is most suitable for objects that are of a uniform, bright colour. The development of a tracking method was described by [14], which only used colour information from an image. The colour tracking method worked by identifying regions of similar average colour in each image frame. The object to be tracked was split into several regions, each of which was described by a colour vector. These regions could then be tracked by evaluating the goodness of fit between a measurement vector and its target. Colour was also used as a feature in [41] to track brightly coloured objects. Since red, green and blue (RGB) colour is very sensitive to illumination changes, [41] used a conversion method which converted RGB colour to the YUV colour space. YUV colour is useful because the intensity component is separated from the colour component which means it can be dealt with separately. The region of the object to be tracked was selected by the user and the colour cluster was determined for that region. This worked well on image sequences where the object's colour was quite distinct from the background. If the object being tracked is very similar in colour to other objects in the background, then it is not an optimal feature to use. Colours should also not be used in scenes with drastic illumination changes, as the colour will vary too much. However, there are ways of overcoming this. A method of object tracking was proposed by [26] which combined colour distributions with particle filtering. The particle filtering allowed for dynamic state estimation, which meant that the colour distributions were updated to allow for changes in lighting. The results from the study by [26] can be seen in Figure 2.1, where the object tracking method was tested on a car.

Edges

Edge information is another popular feature that can be used, which is less sensitive to illumination changes than colour. Edges are also very robust to noise [33]. Edges are

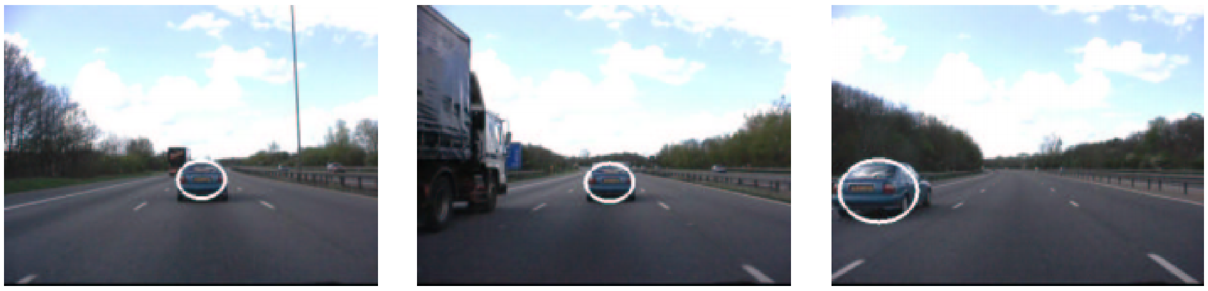

Figure 2.1: Results from the study by [26] where colour distributions and particle filtering were used to track a car.

found where there are large intensity changes in the image. There are many edge detection algorithms and these are generally used with tracking algorithms that track the boundary of objects. Four common edge detection algorithms were compared in [19], including the Canny edge detector, the Sobel edge detector, the Sarkar-Boyer edge detector and the Nalwa-Binford edge detector. The results from the comparison showed that edge detection can be context dependent and each algorithm had better results than the others for certain images. The Canny edge detector, however, is generally acknowledged as the best all-round method for detecting edges [33, p. 271]. It was created by John Canny and proposed in 1986 in the paper entitled “A Computational Approach To Edge Detection”. The main three criteria it satisfies are [33, pp. 271–272]:

- Low error rate so as not to miss important edges
- Edge points found are localized, the distance between them and the real edge is not large
- Minimal response meaning an edge should only be detected once[3]

The Canny filter was used in [41] as part of an object tracking algorithm based on colour thresholding. A function in OpenCV, an open-source computer vision library, was used by [41] to estimate the object’s edges. Figure 2.2 shows the results when the Canny filter was used in [41] to find the edges in a frame, including the edges of the object being tracked, which was a yellow hard-hat.

[3]Matousek et al, Canny Edge Detector, http://www2.it.lut.fi/kurssit/07-08/CT20A6100/seminars/2009-2010/Canny.pdf, 2010

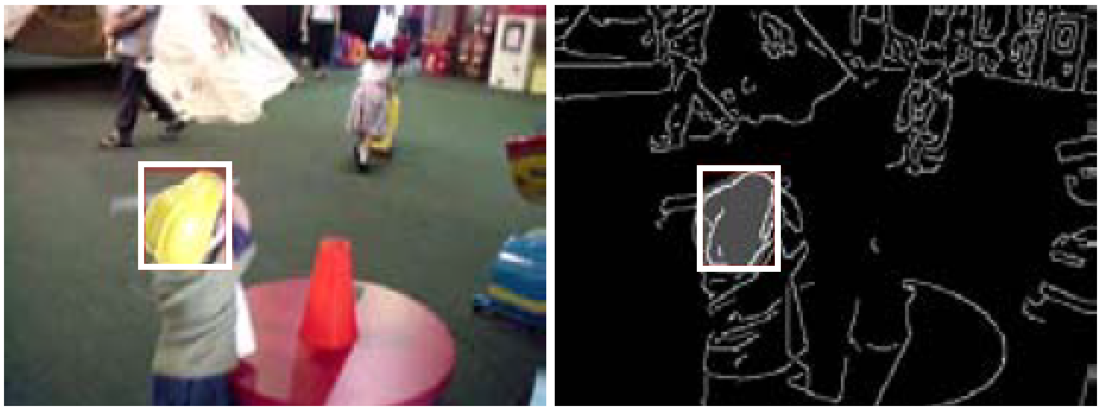

Figure 2.2: Edges found in an image using the Canny filter mask [41].

Texture

Texture is another feature that may be used and it measures properties like smoothness and regularity of an object. This, like edges, is not very sensitive to illumination changes. Texture is a good choice for feature extraction if the texture of the object is very distinct from the background. Operators that may be performed on an image to extract different textures are standard deviation or variance, entropy and local range [33, p. 246]. Some common uses of texture recognition include X-rays, cloud-type recognition and crop yield [35, p. 667]. Texture information is often combined with colour information of an object in order to perform robust object tracking. An object tracking algorithm was proposed in [25] which used a joint colour and texture histogram. This histogram was used with the mean shift algorithm in order to track different objects, including people, cars and faces. The texture features of the object were computed using the local binary pattern (LBP) technique. This technique works by calculating the texture value of each pixel in an image. The texture is represented as a binary pattern. A mask was made in [25] to extract only the pixels that matched the main LBP and colour features. The results of using the technique in [25] on a persons face are shown in Figure 2.3.

2.3.4 Object Detection

Object detection can either be based on temporal data or spatial data [24]. Temporal data usually relies on a static background and movement is therefore equal to the difference between images. Background subtraction is very popular in this case. Temporal data assumes that there is only one moving object in the scene. This means that any differences between images is probably due to the moving object. Spatial data usually

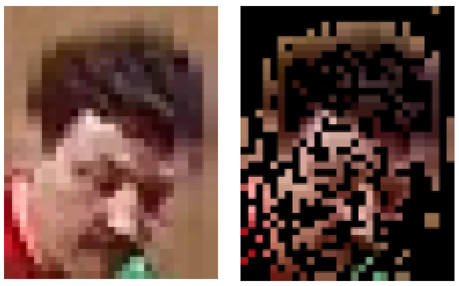
Figure 2.3: The resulting mask using the joint colour and LBP model proposed in [25], used to track a face.

uses thresholding or statistical approaches. Temporal data is generally easier to extract than spatial data [24].

Background Subtraction

Background subtraction is a way of detecting moving objects by finding the difference between the current image and a background image[4]. Background subtraction relies on a fixed camera [7]. Background subtraction can be performed using only two images, but an improved method is to use multiple images to make it more robust [24]. Another method to make background subtraction more robust is to update the background model iteratively so that any changes in the scene are taken into account.

Popular methods of background subtraction were evaluated in [7]. The results of each of these methods can be seen in Figure 2.4 where *Basic* refers to frame difference, *1-G* refers to one-gaussian, *GMM* refers to gaussian mixture model, *KDE* refers to kernel density estimation and *MinMax* refers to minimum, maximum and maximum inter-frame difference.

The simplest form of background subtraction is frame difference, which subtracts the current image frame from the previous image frame. If the difference for a certain pixel is higher than a given threshold, then that pixel is part of the foreground. This method will only work well if the object is moving constantly and if the object is quite distinct from the background [1, pp. 44–45]. Using frame differencing on an object with uniform intensity may result in holes in the foreground mask. These holes will have to be filled with other methods. Frame difference is the fastest of all background subtraction algorithms [1, pp. 44-45]. Frame difference was used as part of the block matching object

[4]Massimo Piccardi, Background subtraction techniques, http://www-staff.it.uts.edu.au/∼massimo/BackgroundSubtractionReview-Piccardi.pdf

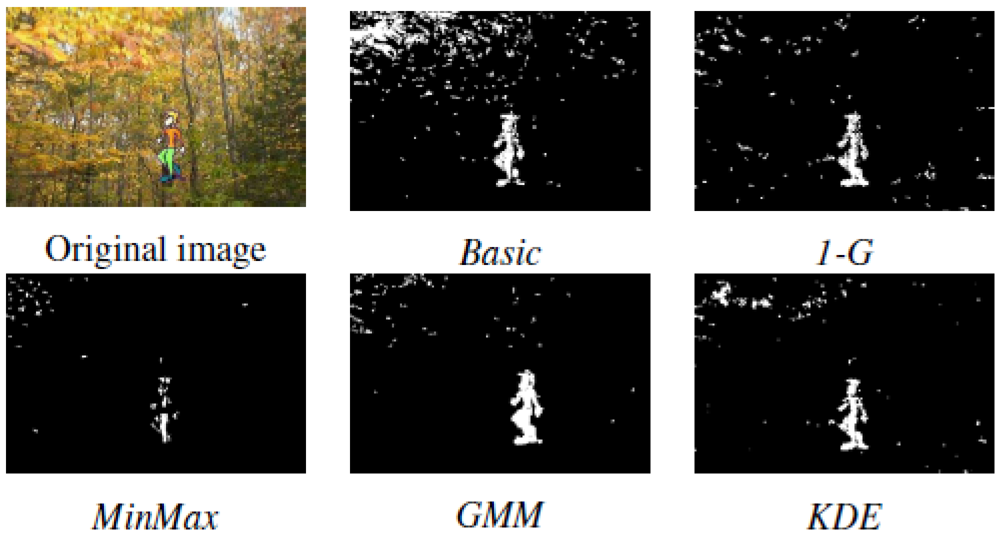

Figure 2.4: Results of different background subtraction algorithms performed on a frame from a multimodal video [7].

tracking algorithm in [37]. Frame difference was used on a low resolution image to detect peoples movements. By using low resolution images, unwanted noise, like moving leaves or branches, was eliminated. Morphological operators can be used to fill holes in the object. The operators used by [37] were dilation and erosion. Another method is to use one Gaussian distribution for every pixel, but this is only sufficient for a scene with static backgrounds [36]. For non-static backgrounds, it is better to use a Gaussian Mixture Model, where every pixel is modelled by a mixture of Gaussians. In [36], multiple Gaussians are used which are updated in order to keep track of changes in the scene such as lighting, or movement in the background.

The results in [7] showed that each method was better suited to a specific scene. For a static background, the *Basic* method, or frame difference, was the most accurate. For noisy backgrounds, the *1-G*, *KDE* and *GMM* were more accurate.

Optical Flow

Flow can also be used for temporal data [24] and this uses points or features in images to detect motion. It analyses the image changes as a result of motion in image sequences [35, pp 685–686]. Optical flow is the velocity of the moving object. The Kalman filter is a popular way of measuring optical flow of a single object. It assumes noise to be white and Gaussian. Since this method also falls under object tracking, it is discussed in more detail later.

Segmentation

Segmentation is used to divide an image into similar parts or regions [42]. Segmentation uses spatial data and can be classified as a statistical approach. This approach is very robust compared to background subtraction and can handle more diverse scenes [24]. Segmentation can be divided into three main categories [35, pp. 123–210]. These are thresholding, edge-based segmentation and region-based segmentation. Thresholding is the fastest and most simple method. Colour thresholding was used by the real-time object tracking algorithm proposed by [41] to find pixels whose colour was above a certain threshold value. Thresholding was also used by [34] to filter out pixels whose colour had not changed very much in two different frames. This meant that only pixels whose colours had changed significantly remained, therefore it could be assumed that there was motion occuring at these pixels. In both cases, thresholding was very sensitive to noise and additional methods, such as low-pass filtering and noise filtering, had to be used to overcome this. Thresholding should only be used if the object has a distinct intensity or colour, compared to the background [24].

The next method is edge-based segmentation which uses edge information, obtained through methods mentioned earlier, to segment an image into objects. Like thresholding, edge-based segmentation is also sensitive to noise [35, p. 207]. This segmentation method is used by the edge detection algorithm presented by [22], whose method was based on a scan line approximation technique. This method was very successful in computing the edges of range images and it was computationally efficient. The last category of segmentation is region-based segmentation. This works by grouping similar neighbour pixels, resulting in the formation of regions. This segmentation method is not as sensitive to noise as the other methods. It is suggested by [35, p. 176], that results from edge and region based segmentation be combined. This is demonstrated in the papers by [28] and [13], where a hybrid of the two segmentation techniques were used for more accurate image segmentation.

2.3.5 Tracking

Tracking combines all the previously mentioned sections to find corresponding objects across image frames. Tracking is used to find the trajectory of an object and its current region at different times in an image sequence [42]. There are three main tracking categories [42]. Point tracking is the first category, where objects are represented by points.

The next category is kernel tracking, which uses a kernel template representing the object's shape and appearance. The motion of this kernel can be computed to track the object. An example of kernel tracking is mean shift. Silhouette tracking is the third category and this estimates an object region containing information such as edge maps and density. Shape matching or contour evolution can then be used to track these regions or silhouettes. Tracking methods can be classified into a bottom-up approach, where an object is extracted from the image and then tracked, or a top-down approach, where an object hypothesis is made and then verified in the image [26].

Point Tracking

In point tracking, objects are represented as points and the motion and position of these points are tracked in consecutive image frames. Point tracking methods can either be deterministic or statistical [42]. Deterministic methods define a cost function which is made up of constraints like maximum velocity, common motion and rigidity [42]. This cost function must then be minimized for tracking. A greedy algorithm can be used for this, which iteratively optimizes point correspondences [40]. The greedy algorithm used by [40], was based on the algorithm used in a paper by Sethi and Jain. The algorithm was modified in [40] to preserve most of the motion information so that point measurements were not missed.

Statistical methods form the other category of point tracking and these model uncertainties to handle noise in an image. A well-known method for statistical point tracking is multiple hypothesis tracking. A set of hypotheses are defined for an object and predictions are made for each hypothesis about the object's position. The hypothesis with the highest prediction is the most likely and is chosen for tracking [42]. Multiple hypothesis tracking was used in [14], in order to overcome occlusion. A set of 10 hypotheses were defined, which could be classed into different occluding states. Probabilities were then assigned to these hypotheses and the likelihoods were established. The results in [14] proved this method to be very successful in the tracking of a head in different states of occlusion.

Other statistical methods that can be used to track single objects are the Kalman filter and Particle filters. The Kalman filter is limited to a linear system and uses prediction and correction to estimate an object's motion [42]. Since the Kalman filter relies on a Gaussian distribution, a particle filter is necessary for state variables that do not follow this distribution. Particle filtering models uncertainty which means that it works well in situations with a lot of clutter or occlusion. Particle filtering was used with colour distributions in the object tracking algorithm introduced in [26]. This algorithm was made for the tracking of non-rigid objects. Initialization of the particle filter was done using an algorithm based on Support Vector Machines. The results from the study in [26], showed that this method of using colour distributions, along with particle filtering, was very effective in tracking fast-moving, non-rigid objects like faces and cars.

Kernel Tracking

Kernel tracking represents an object as a geometric shape, called a kernel, and estimates the motion of this kernel in consecutive frames. Template tracking is commonly used to track a single object. It uses brute force to search an image for a region that matches the template in the previous image [42]. The brute force search results in this method being computationally expensive, but this can be overcome by optimizations to the method such as limiting the search to a certain region. Mean shift was used for template matching in [10], which eliminated the need for brute force. Mean shift was first introduced in 1975 by Fukunaga and Hostetler in the paper entitled “The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition”. It is an iterative algorithm that shifts a point towards the average of other points in that area [9]. Kernel tracking methods can generally be used for real-time tracking, as the rigidity constraint of the kernel allows for fast computation [42]. A limitation of kernel tracking is that parts of the background may appear inside the kernel, but this can be overcome by constraining the kernel to be inside the object, instead of around it [42].

Silhouette Tracking

Silhouette tracking is generally used for objects that cannot be represented by a simple shape and makes use of an object’s appearance. The human body is an example of this, as objects like hands or heads are complex shapes. Silhouette tracking can fall under two categories, shape matching and contour tracking [42]. Shape matching is quite similar to template matching, discussed in kernel tracking, in the way that it searches an image to find a region that matches the silhouette model from the previous image. Shape matching was used for an object recognition framework by [6], where the underlying shape of an object was described by its shape context. The shape context was then used to find corresponding, matching shape contexts in consecutive images. Contour tracking is the other form of silhouette tracking and this differs from shape matching in the way that it uses the contour of an object that evolves in consecutive images. A silhouette based motion tracking system was proposed by [30] to track human motion by using the contour of the body to estimate the pose changes in consecutive images.

2.3.6 Orientation or Pose Estimation

There are many instances when the orientation or pose of the object being tracked needs to be estimated. For example, to recognize different gestures or movements performed by a human body, the configuration of the body needs to be identified. Methods used to recognize these different poses or orientations can be classified into model-free or direct model use [24]. Model free is when there is no a priori information used to estimate the pose. Instead, the pose or orientation can be represented by a set of points or shapes with meaning attached to them. Markers, attached to the object, can also be used in this case. For example for finding the orientation of an object, markers can be placed on it and a vector can be created that measures the distances between the markers[5]. Direct model use involves using a priori knowledge to match a certain pose or orientation to a predefined template [24]. This method requires a large amount of searching, therefore a very efficient search algorithm would need to be used to achieve fast results.

2.3.7 Performance Evaluation of Different Tracking Algorithms

A set of metrics are described in [43] which can be used to evaluate the performance of an object tracking algorithm. These metrics were used to test an industrial tracker from BARCO, which is a global technology company, and the blob tracker from OpenCV 1.0.

The metrics proposed by [43] include high-level ones like True Positive, False Positive and False Negative tracks. True Positive is when a track is correctly detected. False Positive is when there is a false alarm and the track is considered to have been detected when it should not have been detected. False Negative refers to a track detection failure where nothing has been detected. True Negative would be when nothing is detected and there is no track, but since the metrics are only concerned with the object being tracked, it is unnecessary to include it. These metrics can then be used to produce other metrics like specificity and accuracy. Other metrics used are track fragmentation, which shows the lack of continuity of a track, and ID change. These two metrics help to evaluate a track's integrity. The remaining metrics used, help to evaluate the accuracy of the motion

[5]Hersch et al, Iterative Rigid Body Transformation Estimation for Visual 3-D Object Tracking, 2003

tracking itself. Track matching error measures the error between the positions of a track and the ground truth (which means the position of the actual object in real life). This error should ideally be very small. Similarly, closeness of track is a metric that measures the overlap of the system track and the ground truth. Lastly, latency of the system track measures the time between the appearance of an object and when it starts to be tracked.

The two trackers in [43] were then evaluated on a number of different image sequences. Each one had a challenging scene such as illumination changes and fast moving objects. The results show that the BARCO tracker generally had a higher performance than the OpenCV tracker however, the OpenCV tracker was better at estimating an objects position as it had a lower track matching error. The authors argue that without their rich set of metrics, it would be difficult to find the causes of good or bad performance for a given tracker. The authors conclude that these metrics should be used to evaluate more trackers in the future.

The inconclusive results of the study by [43] prove how difficult it can be to measure the performance of an object tracking algorithm. Different algorithms have different strengths and weaknesses and may be very successful in a certain context, but fail completely in another. This means that the tests chosen to measure the success of an object tracking algorithm must not be subjective or biased and must range in context in order to test all aspects of an algorithm.

2.4 Conclusion

The literature discussed shows that image processing and object tracking are both very broad subjects and are made up of a very large number of methods and algorithms. A brief overview is provided of the different implementations that can be used for object tracking and the environment for which they are most suitable. The different categories of autonomous vehicles are also discussed, giving examples from literature to indicate the differences, advantages and disadvantages between them. This information was the main motivation for the decisions made regarding the tools and algorithms used for this project.

Chapter 3

Design and Tools

The general design of the system was based on a simple feedback control loop made up of a camera, the helicopter and an infrared emitter. The Syma S107 infrared mini helicopter was used for this system. It contained an LED on the front and an extra LED was added on its tail. These LEDs were used to track the helicopter. The camera was needed to produce video data of the scene from which the helicopter's position could be extracted. The Kinect for Xbox 360 was chosen as a camera for this system. The Arduino platform was used for the control of the helicopter. The helicopter's movement was controlled by infrared signals sent by an infrared emitter attached to an Arduino Uno board. The logic behind the system required the use of an object tracking library, for which OpenCV was chosen. Certain image processing functions and object tracking algorithms were also needed. Thresholding and CamShift were used for the tracking in the system. The rest of this chapter describes all the system tools mentioned above and the reasons for which they were chosen.

3.1 Software Tools

The following sections describe the software tools used. This includes the theory, libraries and algorithms.

3.1.1 Control Theory

Control theory is used with dynamic systems, which are systems with changing states. Often it is necessary to control these changing systems in order to keep them in some

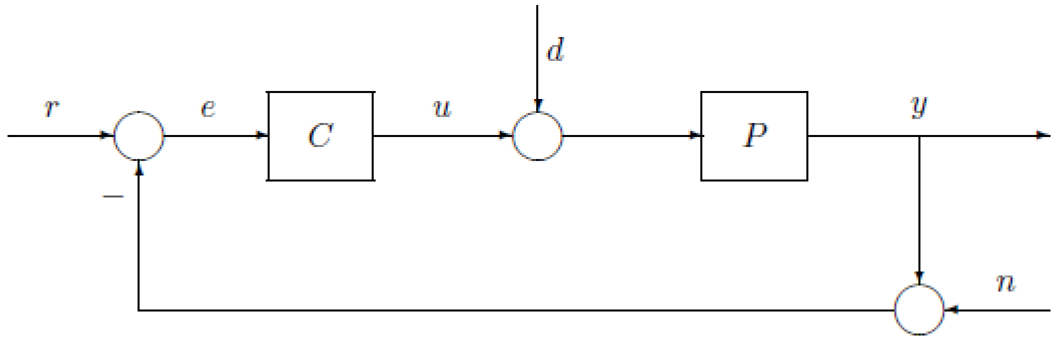

Figure 3.1: Basic Feedback Loop [11, p. 8].

stable state and prevent oscillation. An open loop controller sets the system at some desired state, but does not take changes into account which could at some point alter the state of the system [31]. A closed loop controller, however, takes into account the changes in the system by using feedback and adjusts the state of the system in order to change it back to the desired state. In order to do this, the inputs and outputs of the systems need to be observed. If the outputs are unsatisfactory, then the inputs must be corrected in order to overcome the error produced in the outputs [31]. The basic feedback control loop is shown in Figure 3.1 [11, p. 8].

- r - reference or command input
- e - tracking error
- u - control signal, controller output
- d - plant disturbance
- y - plant output
- n - sensor noise

Figure 3.1 shows a closed loop controller based on negative feedback. The controller C, is responsible for controlling the system to keep it at some desired state. The plant P, is the dynamic system that needs to be controlled. The term “negative feedback” implies that the controller analyses the error e between the reference or desired input r, and the system’s current output y. The controller then counteracts this error by changing the signal u which will in turn affect the output y. The output is obtained from feedback produced by a sensor. This sensor may produce some noise n which could obscure the inputs received and may need to be compensated for. The error that may occur would be produced by some disturbance d which alters the state of the plant, changing the output [11, pp. 31–32]. A closed loop controller was used for the system as feedback was necessary to adjust the helicopter's movements depending on its position. If an open loop controller was used, where no feedback was acquired, the helicopter's movements would have had to be set beforehand. This would mean that no adjustments could be made to account for changing conditions, such as gusts of wind generated from the rotors of the helicopter.

3.1.2 OpenCV

The Open Source Computer Vision library was chosen to provide the image processing and object tracking functions for the system. It is a library containing real-time computer vision programming functions which are based on important image processing algorithms. The reason OpenCV was chosen for this system is because the functions are optimized and therefore suitable for real time applications. Many of these algorithms can be used for the tracking of objects. OpenCV was originally written in C but has now been shifted to C++ and has a full C++ interface. The C++ interface was used for this system. OpenCV provides abstraction with a high-level API, but also allows for technical programming with a low level API.

In OpenCV, an image can be represented by a matrix which holds all the image's pixel values. OpenCV contains many pre-defined matrix operations which simplify the handling of image data. For this reason, matrices were used to represent the images in this system. Full documentation for OpenCV can be found online[1].

3.1.3 Thresholding

Thresholding is a point operation. A point operation is a simple segmentation method where a pixel's original value is the only thing that determines what its new value will be. Thresholding uses a threshold value which is compared with a pixel's original value and depending on the type of thresholding, the pixels new value is calculated. OpenCV defines five different types of thresholding [27].

[1]OpenCV Wiki, http://opencv.willowgarage.com/wiki/

Binary Thresholding

The first type of thresholding is called binary thresholding. The resulting image is a black and white binary image. The formula for binary thresholding is as follows:

$$dst(x,y) = \begin{cases} maxVal & \text{if } src(x,y) > thresh \\ 0 & \text{otherwise} \end{cases}$$

where $thresh$ is the threshold value, $src(x,y)$ is the pixel's value and $maxVal$ is the maximum pixel value [27]. If the pixel's value is higher than the threshold, it is set to the maximum value whereas if the pixel's value is lower than the threshold, it is set to 0.

Inverted Binary Thresholding

The second type of thresholding is called inverted binary thresholding. The resulting image is a black and white binary image. The formula for inverted binary thresholding is as follows:

$$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > thresh \\ maxVal & \text{otherwise} \end{cases}$$

where $thresh$ is the threshold value, $src(x,y)$ is the pixel's value and $maxVal$ is the maximum pixel value [27]. If the pixel's value is higher than the threshold, it is set to 0 whereas if the pixel's value is lower than the threshold, it is set to the maximum value.

Truncated Thresholding

The third type of thresholding is called truncated thresholding. The resulting image is the same type as the original image. The formula for truncated thresholding is as follows:

$$dst(x,y) = \begin{cases} thresh & \text{if } src(x,y) > thresh \\ src(x,y) & \text{otherwise} \end{cases}$$

where $thresh$ is the threshold value and $src(x, y)$ is the pixel's value [27]. If the pixel's value is higher than the threshold, it is set to the threshold value, whereas if the pixel's value is lower than the threshold, it keeps its original value.

Thresholding To Zero

The fourth type of thresholding is called thresholding to zero. The resulting image is the same type as the original image. The formula for thresholding to zero is as follows:

$$dst(x, y) = \begin{cases} src(x, y) & \text{if } src(x, y) > thresh \\ 0 & \text{otherwise} \end{cases}$$

where $thresh$ is the threshold value and $src(x, y)$ is the pixel's value [27]. If the pixel's value is higher than the threshold, it keeps its original value, whereas if the pixel's value is lower than the threshold, it is set to 0.

Inverted Thresholding To Zero

The fifth type of thresholding is called inverted thresholding to zero. The resulting image is the same type as the original image. The formula for inverted thresholding to zero is as follows:

$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > thresh \\ src(x, y) & \text{otherwise} \end{cases}$$

where $thresh$ is the threshold value and $src(x, y)$ is the pixel's value [27]. If the pixel's value is higher than the threshold, it is set to 0 whereas if the pixel's value is lower than the threshold, it keeps its original value.

The reason thresholding was used for segmentation over the other methods discussed in Chapter 2, is because it is extremely fast and computationally efficient. This high performance is necessary for a real-time system. A common use of thresholding is to create a mask for an image which can be used to extract an object or a region of interest from the image. Binary thresholding was used for the system in order to create a mask that contained only the brightest pixels in the images. If a pixel's brightness value was higher than the threshold, it was set to white otherwise it was set to black. The resulting mask therefore contained the location of the two LEDs on the helicopter.

3.1.4 CamShift

The CamShift algorithm (Continuously Adaptive Mean Shift) is a colour-based tracking algorithm and is an adaptation of the mean shift algorithm.

Mean Shift Algorithm

The mean shift algorithm was first proposed in a paper by Fukunaga and Hostetler [15], as mentioned in Chapter 2. This algorithm can find an object by its colour histogram. It uses the gradient of a probability distribution to locate the peak of that distribution [2]. The basic steps of the mean shift algorithm are shown [21] :

1. Set the search window dimensions
2. Select the area of the frame in which the search window should be located
3. Calculate the mean location (centroid) of the coloured object inside the window
4. Use this location as a centre point for the search window
5. Repeat from step 3 to step 4 until the centre of the search window converges

The centre point calculated in the algorithm is found using moments. In image processing, a moment is a projection of an image's function. Moments are often used in image processing for pattern recognition and for calculating statistical properties of an image, such as the area or the centroid[2]. The zeroth moment is first found by using:

$$M_{00} = \sum_x \sum_y I(x, y).$$

The first moment of x and y is then found, using:

[2]Jamie Shutler, Statistical moments, http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SHUTLER3/node1.html, 2002

$$M_{10} = \sum_x \sum_y x I(x,y),$$
$$M_{01} = \sum_x \sum_y y I(x,y).$$

The centroid can then be found, using:

$$x_c = \frac{M_{10}}{M_{00}},$$
$$y_c = \frac{M_{01}}{M_{00}},$$

where x and y range over the entire search window and $I(x,y)$ is the probability value of that pixel at (x,y) [21].

These probability distributions are static, meaning they are not updated regularly. This means that mean shift would not work very well with dynamically changing distributions, such as in a video.

CamShift Algorithm

The CamShift algorithm is an adaptation of mean shift, which allows the handling of dynamic distributions. It iterates toward the maximum increase in probability density to find the mean, using spatial moments. CamShift allows for adaptive distributions, which are updated for each frame [2]. Every frame gets converted into a new probability distribution. CamShift was originally intended for human face tracking. Figure 3.2 shows the basic CamShift algorithm. The grey area shows the mean shift algorithm that CamShift is based on. The basic CamShift algorithm is [21]:

1. Select the whole frame to be used for the calculation of the probability distribution.
2. Select the area in the frame which will determine the co-ordinates of the initial search window.
3. Calculate the colour probability distribution in a window slightly bigger than the mean shift window size.

4. Find the search window centre and zeroth moment using mean shift.
5. In the next frame, use the centre location from the mean shift and centre the search window around it. Change the search window size to a value found using a function of the zeroth moment.

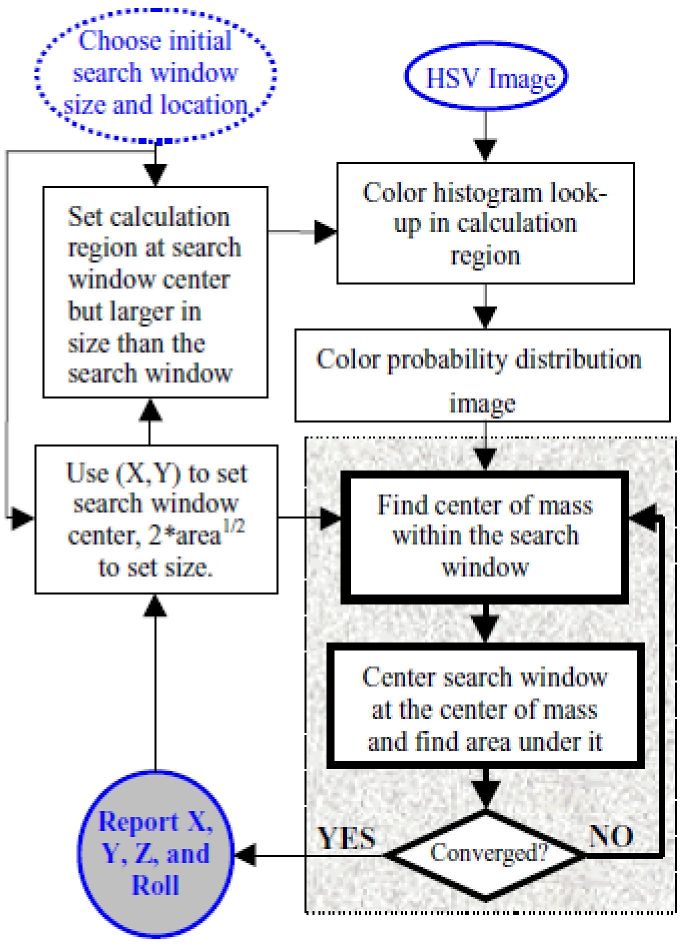
Choose initial search window size and location
HSV Image
Set calculation region at search window center but larger in size than the search window
Color histogram look-up in calculation region
Color probability distribution image
Use (X,Y) to set search window center, 2*area^1/2 to set size.
Find center of mass within the search window
Center search window at the center of mass and find area under it
YES
Converged?
NO
Report X, Y, Z, and Roll

Figure 3.2: The basic CamShift algorithm [21].

The new size of the search window can be calculated as follows [12], [21]:

The second order moments are first calculated by using:

$$M_{20} = \sum_x \sum_y x^2 I(x,y),$$
$$M_{02} = \sum_x \sum_y y^2 I(x,y),$$
$$M_{11} = \sum_x \sum_y xyI(x,y).$$

Then a, b and c are calculated by using:

$$a = \frac{M_{20}}{M_{00}} - x_c^2,$$
$$b = 2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right),$$
$$c = \frac{M_{02}}{M_{00}} - y_c^2.$$

The length l and width w of the new search window can then be found, using:

$$l = \sqrt{\frac{(a+c) + \sqrt{b^2 + (a-c)^2}}{2}},$$
$$w = \sqrt{\frac{(a+c) - \sqrt{b^2 + (a-c)^2}}{2}}.$$

CamShift was used as a tracking method for this system because it is computationally efficient, resulting in fast performance [20]. These qualities make it suitable for real time object tracking. CamShift was also suitable for the system as the helicopter was bright red in colour and therefore could be tracked easily by CamShift.

3.1.5 Arduino Environment

Arduino provides downloadable software[3] to be used with an Arduino board. The environment is mainly based on Processing and is written in Java. It is very user friendly

[3]Arduino Home Page, http://www.arduino.cc/

and aims to allow users to quickly and easily write code to program the Arduino board. The environment provides a GUI in which to write, compile and upload the code. This environment was used for the system as it is compatible with the Arduino board, which is discussed in the Hardware section.

3.2 Hardware Tools

The following sections describe the hardware tools used. These include the camera, the helicopter and the microcontroller.

3.2.1 Helicopter

The Syma S107 infrared remote controlled helicopter was chosen for this project. This is a 3 channel coaxial helicopter with two stacked rotors and a stabilizer bar for extra stability on top. It also has a built in gyroscope which helps to maintain the helicopter's orientation. It has three basic movements which are the yaw (the rotation), the pitch (forward or backward movement) and the throttle. The trim of the helicopter can be adjusted on the remote in order to stop it from rotating or drifting on its own. The range of the helicopter is roughly ten meters. The helicopter is only suitable for indoor flying because is uses infrared communication. The flight time is about 5 - 10 minutes per charge and the charge time is about 30 minutes. The helicopter is charged by USB or by the batteries in the remote. The two rotors on the helicopter have a fixed pitch which means the rotors have a fixed angle. If the throttle is increased, the rotors rotate at a higher speed and if the throttle is decreased, the rotors rotate at a lower speed. This slowing down and speeding up requires a small amount of time in order to overcome the inertia or the stored kinetic energy and can therefore cause a slight delay or lag between movements[4].

The two main reasons this particular model was chosen were its low price and stable flying due to the gyroscope. It would have been very difficult to implement the system with a helicopter that flew erratically, with little stability. Many other mini helicopters may have been suitable for the system, but this was of little importance since the focus of the system was not on the helicopter but on the software controlling it.

[4]RC Helicopter Fun, John Salt, http://www.rchelicopterfun.com/collective-pitch.html

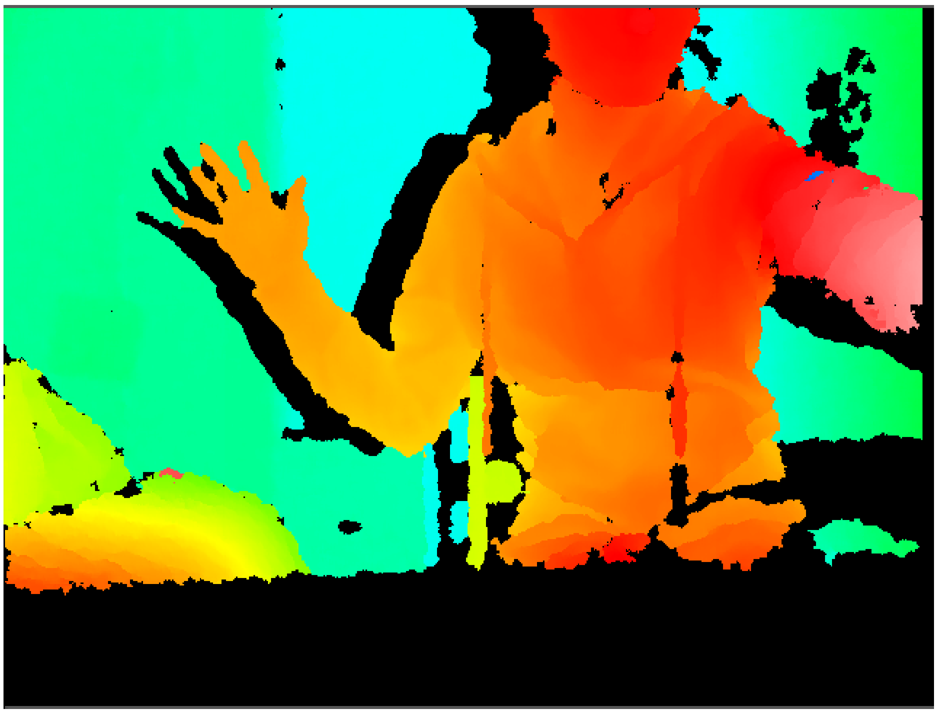

Figure 3.3: The Kinect depth sensor output.

3.2.2 Kinect

The Microsoft Kinect for Xbox 360 was chosen for this project. This is a motion sensing input device and was used for the vision in the project. It contains an RGB camera which produces colour video and it also has a depth sensor which is made up of an infrared laser projector, as well as a monochrome CMOS sensor. The depth map produced by the depth sensor can be seen in Figure 3.3. Each shade represents a certain distance of the objects in the scene. Black indicates that no depth value could be found at that position. The shadows produced in the depth map in Figure 3.3, are a result of the alignment of the laser projector and the sensor inside the Kinect camera. The laser projects an infrared pattern onto the objects in the scene and these objects result in the infrared getting blocked at that point [5]. This means that the sensor, which is to the left of the laser, will not pick up any infrared behind the objects, resulting in no depth values being produced and therefore a black shadow will occur. The sensor can only find depth values inside the laser's line of sight [5]. Figure 3.4 illustrates this shadowing process.

The Kinect's horizontal field of view is 57 degrees and its vertical field of view is 43 degrees. The RGB camera has a resolution of 640x480 and a frame rate of 30FPS [5]. The depth sensor range is from 800mm to 4000mm. The Kinect also contains four microphones, but audio was not used in the system. The Kinect should be used in a well lit environment,

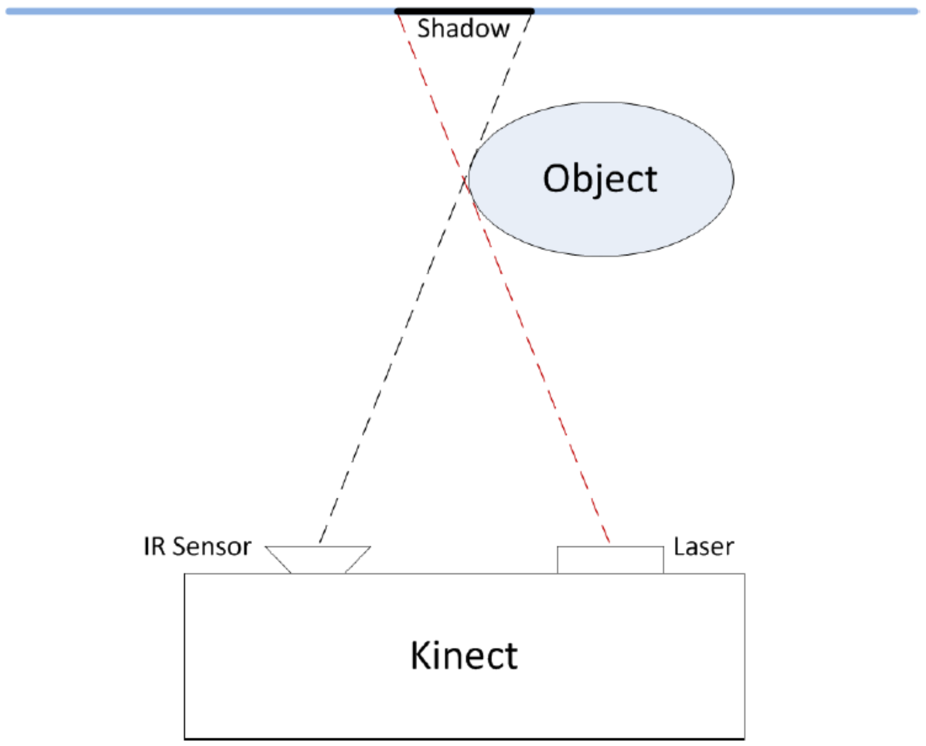

Figure 3.4: An illustration of the shadows that appear in the Kinect depth map due to the alignment of the laser and the sensor [5].

but since it uses infrared it cannot be exposed to sunlight. Drivers and libraries necessary for the Kinect to be used with a computer were obtained from OpenKinect[5]. OpenKinect is a project that focuses on building libraries which allow integration of the Kinect with devices other than the Xbox.

The reason the Kinect camera was chosen for this project, was the fact that it allows for 3D tracking instead of the 2D tracking resulting from normal web cameras. The depth sensor combined with the RGB camera meant that x, y and z co-ordinates of an object could be found. Since a helicopter makes use of all three of these vertices, this 3D tracking was necessary. Since the Kinect is a rather large camera and the Syma S107 is a very small helicopter, the intelligence of the system had to be external, as discussed in Chapter 2.

[5]OpenKinect Home Page, http://openkinect.org/wiki/Main_Page

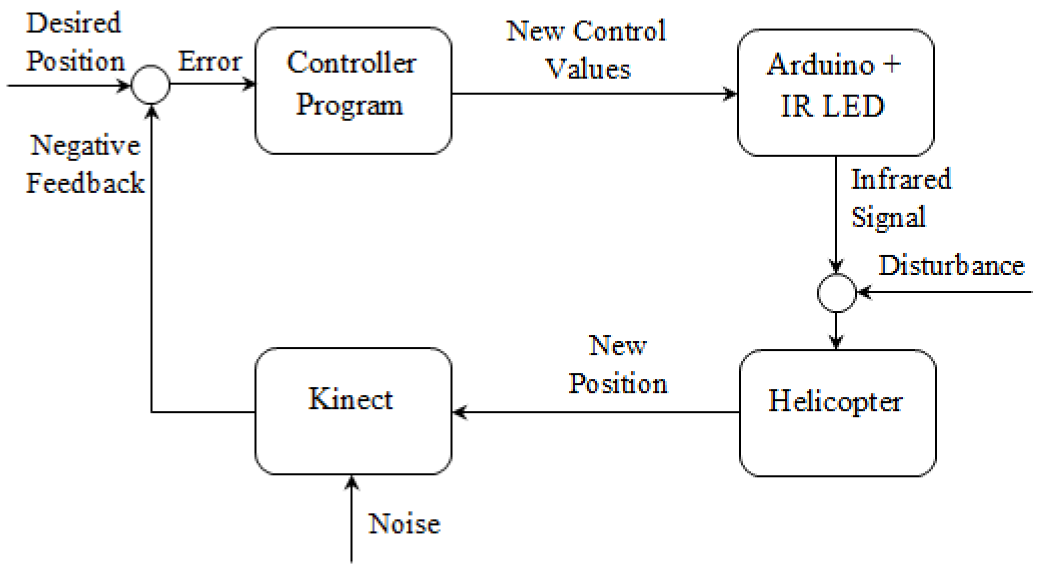

Figure 3.5: Basic feedback loop for the auto-pilot system.

3.2.3 Arduino Uno Board

An Arduino Uno is a micro-controller based on the ATmega328, which is a high performance micro-controller created by Atmel. It consists of digital input and output pins and analogue inputs which can be used to create an interactive environment. It can connect to a computer via USB. An Arduino Uno was used along with an infrared LED in order to control the helicopter from the computer. This effectively replaced the original remote controller that came with the helicopter. This board was chosen for the system because of its low price and the fact that it is very easy to use and program.

3.3 Summary

Figure 3.5 shows the feedback loop specific to the auto-pilot system. The infrared LED connected to the Arduino Uno acted as the remote controller for the helicopter. The plant was the moving helicopter whose position changed often and its position could be changed by disturbances such as gusts of wind resulting from the movement of the propellers or the environment. The Kinect camera produced frames which represented the output of the system. The position of the helicopter was extracted from these frames. The noise produced by the camera may have distorted the image in certain ways which could have obscured the position of the helicopter. The position of the helicopter was then compared with the desired position. The difference between the two was considered the error. The error was counteracted by the program acting as the controller, by changing the control values of the helicopter. The control values were translated to infrared signals which were transmitted by the infrared LED connected to the Arduino. This resulted in the helicopter's movements being changed in order to move it to the desired position.

Chapter 4

Implementation

The implementation of the system was divided into three phases, namely control, tracking and combined tracking and control. The control phase consisted of reverse engineering the infrared protocol of the Syma S107 helicopter in order to control the helicopter from the infrared emitter connected to the Arduino. A high power infrared LED was used to send the infrared pulses. The tracking phase consisted of determining the helicopter's position by using image processing and object tracking functions. These functions could be used on the frames and depth map produced by the Kinect, in order to extract and track the x, y and z co-ordinates of the helicopter. LEDs attached to the helicopter were used to identify it in each frame. These tracking and control phases were then combined in order to create a closed loop controller which could control the helicopter autonomously.

4.1 Control

The maximum power infrared LED kit was assembled according to the schematic in Figure 4.1. The power was connected to the 3.3V output, the ground was connected to the ground pin and the input was connected to digital output pin 12 on the Arduino Uno. The Arduino Uno was connected to the computer using a USB cable. The finished assembly can be seen in Figure 4.2.

4.1.1 Reverse engineering the infrared signals

The signals sent by the original remote were monitored by connecting an oscilloscope across the infrared LED. Different movements were performed with the original remote

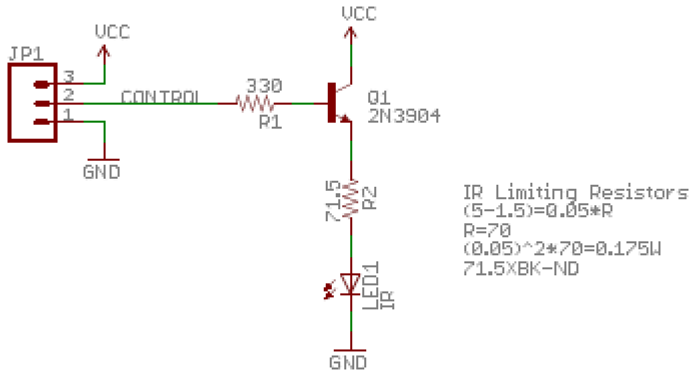

Figure 4.1: Schematic drawing of maximum power infrared LED kit.

and the resulting infrared signals captured on the oscilloscope screen were recorded. In Figure 4.3, one full infrared packet can be seen.

PWM

An infrared packet consists of many, quick pulses of infrared light. These pulses are the result of Pulse Width Modulation (PWM) carrier pulsing. An "on" is represented by many on/off pulses of the infrared LED. This pulsing is performed at a certain rate, called the carrier rate. An infrared detector can be set to detect a certain frequency, which means that it will not pick up infrared signals sent at different frequencies, allowing for multiple devices to operate in the same area. This means that PWM also reduces the effects of ambient lighting. Another reason for PWM is to allow the LED time to cool off, as it uses a significant amount of current and should therefore only be on for a few milliseconds at a time[1]. It can be seen in Figure 4.4 that the carrier frequency for the Syma S107, shown in the Delta section, is approximately 38kHz.

Infrared Packet Structure

The packet in Figure 4.3 is made up of three sections. The first section is called the header. The header consisted of two bits. The first bit was an "on" of about 2000ms in length and the second bit was an "off" of about 2000ms. This header signalled to the infrared detector that the data was on its way. Figure 4.5(a) shows the full header signal. The next section is the data section which consisted of 32 bits. These 32 bits made up

[1]Barry Gordon, Infra Red Signals, http://www.hifi-remote.com/infrared/IR-PWM.shtml

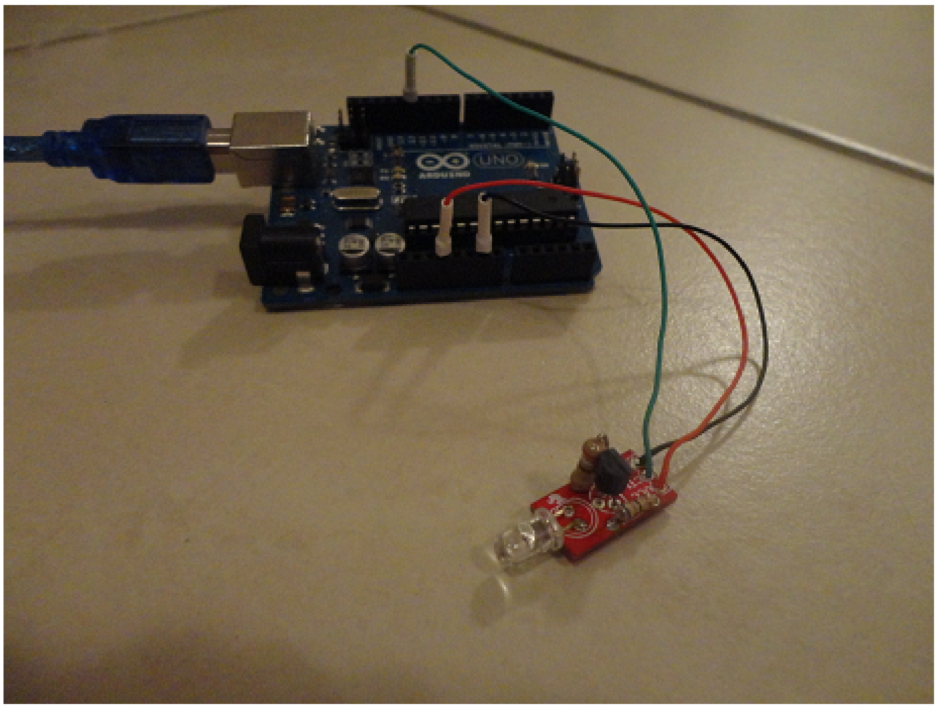

Figure 4.2: Assembled Max Power IR LED kit connected to the Arduino Uno.

four, 8 bit numbers. The first 8 bits symbolized the yaw, the second 8 bits symbolized the pitch, the third 8 bits symbolized the throttle and the fourth 8 bits symbolized the trim.

The first bit of each 8 bit number was always set to 0 which meant that the maximum number for all of the controls was 127. The throttle, however, was an exception as the first bit symbolized the channel that the helicopter was flying on. If the first bit was a 0, it indicated that the helicopter was flying on channel one and if it was a 1, it indicated that the helicopter was flying on channel two. The yaw, pitch and trim had a default value of 63, as they needed to be centered. To make the helicopter turn right, the yaw and trim were decreased and to make it turn left, the yaw and trim were increased. To make the helicopter fly forward, the pitch was decreased and to make it fly backward, the pitch was increased. The throttle had a default value of 0. In order to make the helicopter fly upwards, the throttle and to be increased and to make it move downwards, the throttle had to be decreased.

Each bit (0 or 1) consisted of a pair of on/off infrared pulses. A 1 in the data section was symbolized by sending a short "on" of about 300ms in length, followed by a long "off" of about 700ms. Figure 4.6(b) shows a 1 on an oscilloscope. A 0 in the data section was symbolized by sending a short "on" of about 300ms in length, followed by a short "off"

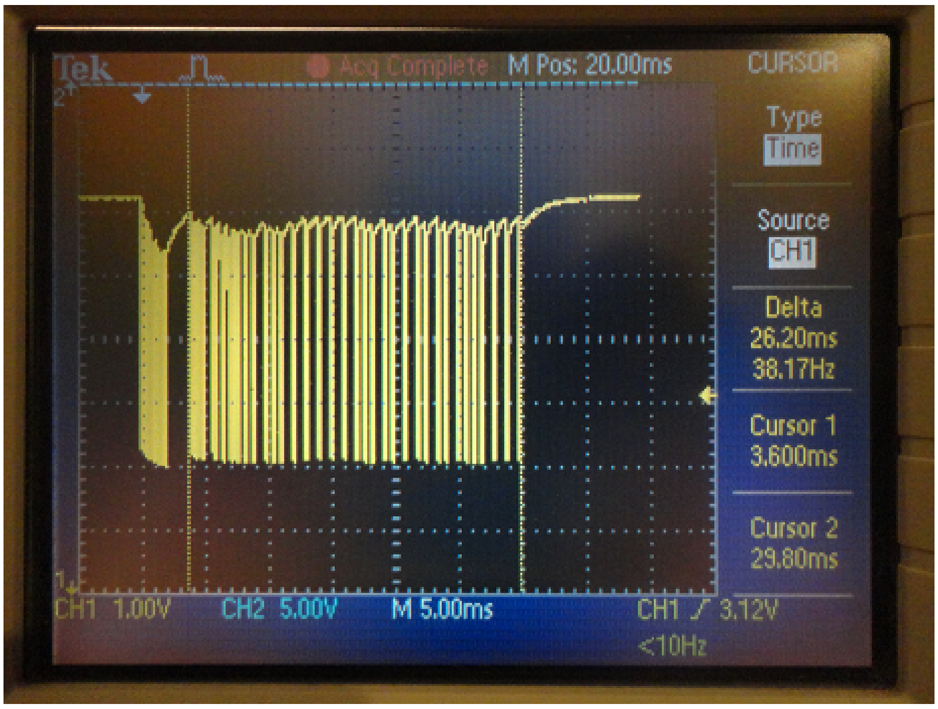

Figure 4.3: A full infrared packet shown on an oscilloscope.

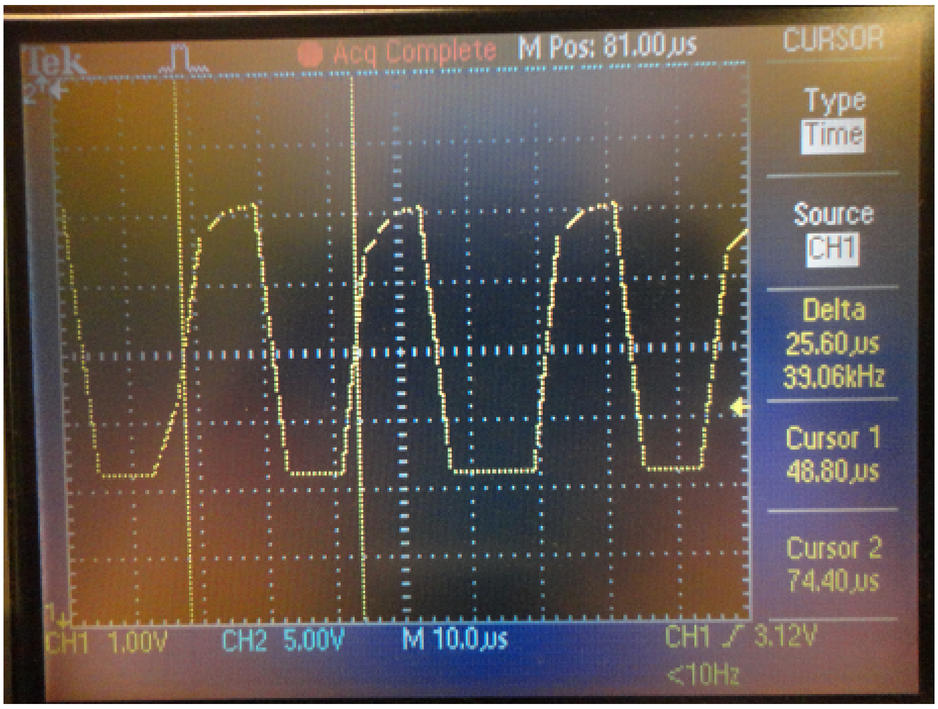

Figure 4.4: The carrier frequency shown on an oscilloscope.

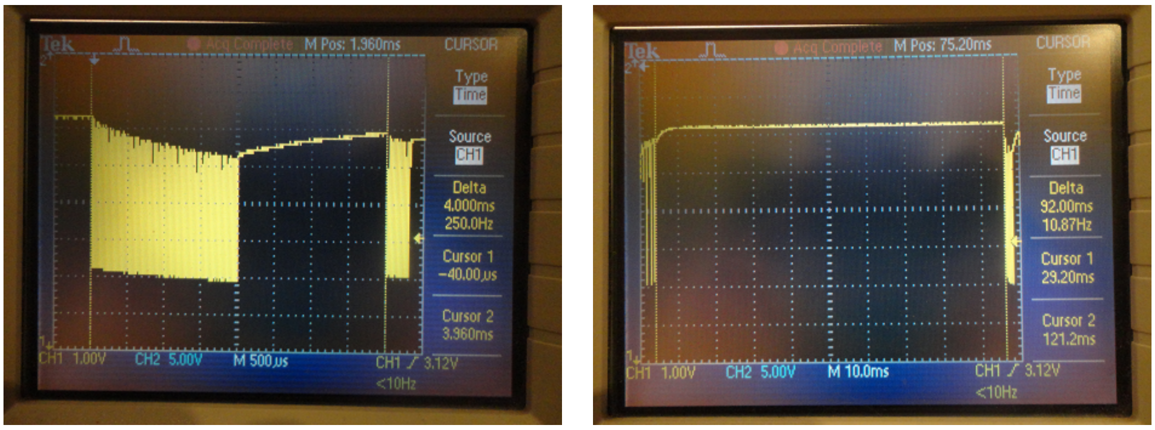

Figure 4.5: (a) The on/off signal symbolizing the header signal shown on an oscilloscope (b) The on/off signal symbolizing the footer signal shown on an oscilloscope.

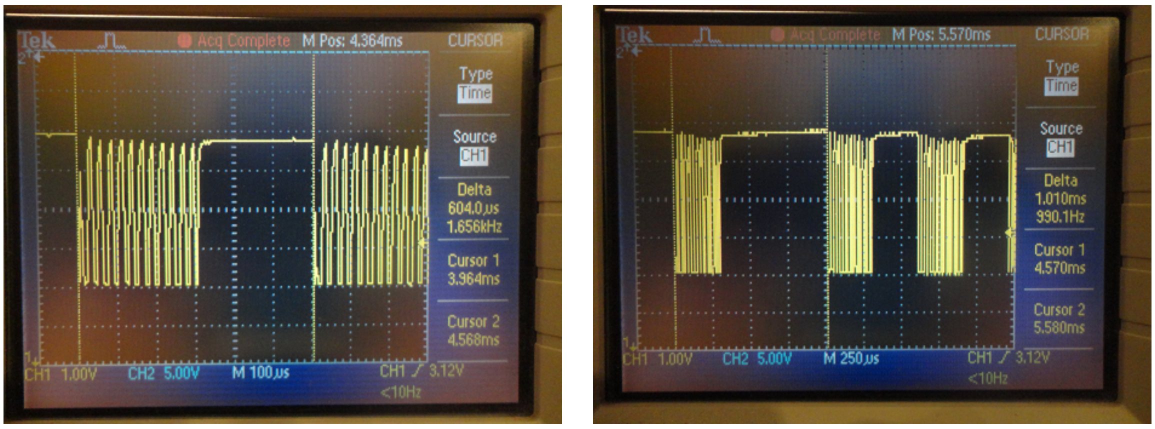

Figure 4.6: (a) The on/off signal symbolizing a 0 shown on an oscilloscope (b) The on/off signal symbolizing a 1 shown on an oscilloscope.

of about 300ms. Figure 4.6(a) shows a 0 on an oscilloscope. The last section is called the footer which was made up of 2 bits. The first bit was an "on" of about 270ms in length and the second bit was an "off" and its length varied in order to keep the whole infrared packet's duration the same as the others. For example, if there were many 0s in the data section of one packet, its duration would be much less than a packet containing many 1s in its data section. The footer of the shorter packet would therefore be longer than the footer of the second packet. Figure 4.5(b) shows the footer signal for a certain infrared packet. The full infrared packets were sent every 120ms from the original remote controller. A simple example of a full infrared packet of the Syma S107 infrared protocol can be seen in Figure 4.7.

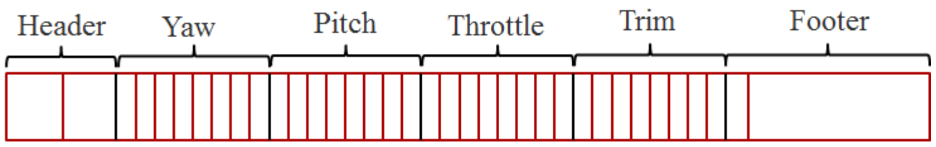

Figure 4.7: An example of a full infrared packet of the Syma S107 infrared protocol.

4.1.2 Sending the Infrared Signals

In order to control the helicopter from the computer, a stream of these infrared packets needed to be sent out, with varying data values for different movements. For example, if the throttle needed to be increased, the third set of 8 bits could be changed in order to increase the current throttle. If the throttle was at 20 and it needed to be increased to 40, the third set of 8 bits would change from 00010100 to 00101000.

Since the infrared signals needed to be sent via the infrared LED attached to the Arduino input, it was necessary to create a program which could be uploaded on to the Arduino, which would control the infrared LED. The program used was based on one found in a forum at RCGroups[2]. The program was altered to allow for changes sent via serial communication. The program was written in the language created by the Arduino team and was compiled and uploaded using the IDE that was downloaded from the Arduino website. The program needed to specify the data to be sent, how this data should be encoded into 1s and 0s and how these 1s and 0s were to be sent by the infrared LED.

The Baud rate was set to 9600 as this is the rate that the Arduino board uses. A method was created to specify how to send a 1 for the data section. This method consisted of turning the infrared LED on for 300ms and then turning it off for 700ms. Similarly, a method was created to specify how to send a 0 for the data section. This method consisted of turning the infrared LED on for 300ms and then turning it off for 300ms. Methods were also created specifying how to send the header and the footer, following the same pattern as the other methods.

Since this infrared protocol used PWM, each infrared pulse (or “on”) had to be made up of many on/off infrared pairs happening at the carrier rate, which was measured previously on an oscilloscope. A method was created to implement this which estimated 38 kHz to be about 13μs on and 13μs off. It took approximately 3μs to send the digital write commands to the Arduino, therefore a digital write command to turn on the LED was

[2]RCGroups, http://www.rcgroups.com/forums/showthread.php?t=1417249&page=4, 2011

sent, followed by a delay of 10μs. This was then repeated to turn the LED off. These on/off pairs were then looped for the specified duration of the "on".

Since the Arduino uses serial communication to listen for incoming data, it was necessary to create a method to handle incoming serial data and deal with it appropriately. In this case, the incoming serial data consisted of changes in the infrared data values corresponding to changes in the yaw, pitch, throttle or trim values. The buffer holding the packet information was then set accordingly and these values were converted in to binary numbers to be sent. The core functionality of the Arduino program consisted of the infinite loop that checked for any changes in the packet data, sent the header, sent the binary data values and then sent the footer. This loop had to continue for the duration that the Arduino was plugged in, as the helicopter required a constant stream of infrared packets.

4.1.3 Controlling the Helicopter from the Computer

To test the functionality of the control program, another program was written in C++ which sent serial data through to the control program running on the Arduino in order to change the movements of the helicopter. This serial data was obtained by creating a window filled with eight buttons, two for increasing or decreasing the throttle, two for moving the helicopter forward or backward, two for moving the helicopter left or right and two for changing the trim left or right. When a button was pushed by the user, the value of that specific control was changed and this change was sent serially to the Arduino program which in turn sent the infrared commands which controlled the helicopter. Since the infrared packets were sent every 120ms from the original controller, a delay of 120ms was needed in order to prevent the system from sending the infrared signals too quickly. The time for the data to be sent serially to the Arduino was approximately 2ms. This was measured by sending commands from the program, increasing the throttle to its maximum and then decreasing it back to zero. The time for all of these commands to be sent to the Arduino was calculated and divided by the number of commands. This 2ms delay meant that a delay of only 118ms had to be added before the data was sent to the Arduino. This delay did not take into account the time taken to process each frame from the Kinect for the tracking of the helicopter, discussed in the next section, after which it had to be decreased to 90ms. The timing information for the tracking methods is discussed in Chapter 5. Using this program, the helicopter could be controlled and flown in the same way as it would with its original remote.

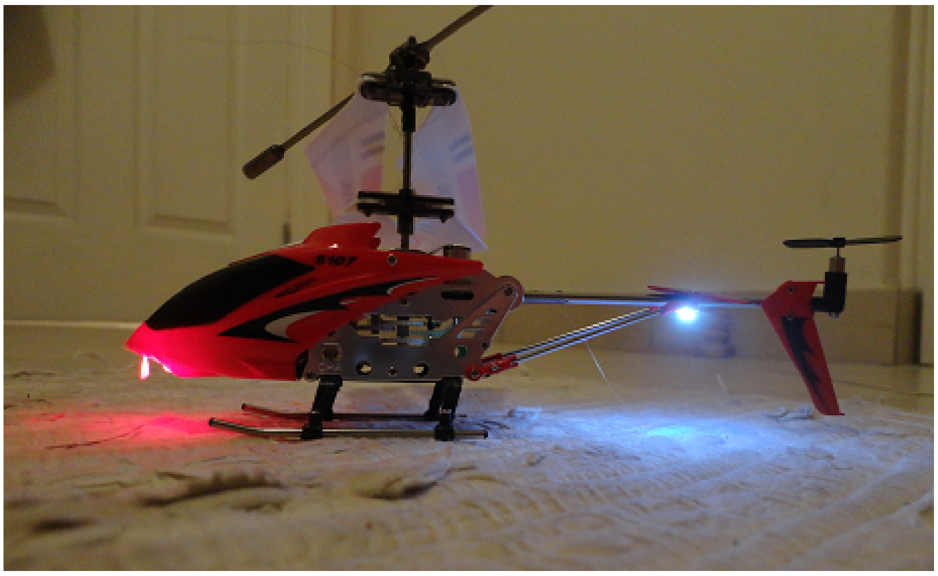

Figure 4.8: Modified Syma S107 helicopter with LEDs.

4.2 Tracking

The Syma S107 helicopter came with a flashing red and blue LED attached to its front end. In order to use this LED as a point for tracking, it was necessary to alter it so that it stopped flashing and shone red constantly. Since the front LED was placed quite far into the body of the helicopter, it was necessary to stick a small square of tape in front of it to reflect it and make it brighter. A plain white LED was attached to the tail of the helicopter so that its back end could be located more easily. This LED was connected to the helicopters circuit board. Figure 4.8 shows the modifications.

4.2.1 Extracting the Helicopter's Position

The x and y co-ordinates of the helicopter were obtained from the image frames captured by the RGB camera of the Kinect. These frames were analysed and processed one by one using OpenCV library functions, in order to extract the helicopter's co-ordinates in each one. The frames captured were put into OpenCV matrices which could then be processed. The helicopter was tracked by finding the positions of the two LEDs attached to it. A binary thresholding function was used to extract only pixels of a certain brightness in the frame. Since it is difficult to measure brightness using the RGB colour space, it was necessary to convert the images into the HSV colour space.

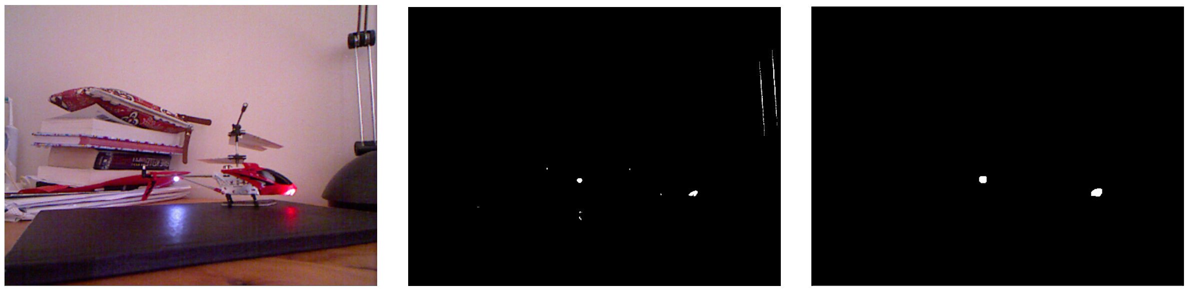

Figure 4.9: (a) The original frame (b) The mask produced by thresholding containing noise (c) The mask produced by thresholding after erosion and dilation.

Thresholding

The Hue, Saturation and Value (HSV) colour space made it much easier to measure brightness. The hue is the actual colour or combination of colours, the saturation measures the colourfulness relative to brightness and the value measures the amount of light emitted. Value could therefore be used to judge how bright a certain pixel was, and if the value was very high it could be assumed that the pixel was very light. This meant that a thresholding function could be used which used a threshold of a certain brightness value. Any pixel lower than this threshold was coloured black and any pixel higher than this threshold was coloured white. The OpenCV threshold function was implemented which resulted in a mask showing the brightest areas in the image. These areas corresponded to the LEDs on the helicopter. In order to remove any noise created by unwanted bright pixels, the mask was eroded and then dilated to enhance the wanted pixels. Figure 4.9(a) shows the original frame used for thresholding. Figure 4.9(b) shows the mask produced using thresholding, which contains small amounts of noise. Figure 4.9(c) shows the mask after it has been eroded and dilated in order to remove the noise.

The threshold value was originally constant and was obtained through experimentation. However, the perceived brightness of the LEDs decreased the further the helicopter moved away from the camera, which meant that the threshold value had to be small enough to include the LEDs at all distances. This meant that the mask often included unwanted areas whose brightness value was larger than the threshold value. In order to solve this problem, the threshold value was made dynamic. The distance of the helicopter from the camera was calculated, using the depth map produced by the Kinect. The distance was used to determine how large the threshold value needed to be. If the helicopter was close to the camera, the threshold value was increased. If the helicopter was far away from the camera, the threshold value was decreased.

Blob Tracking

The OpenCV blob library was used to label each white area in the mask, corresponding to each LED, as a blob. If the helicopter was facing straight into the camera or straight away from the camera, there was only one blob, as only one LED would be visible. If the helicopter was at any other orientation, there were two blobs as both the LEDs would be visible. Once the blobs were created, the blob library was used to calculate certain features of each blob, such as its area, its minimum and maximum x and y values and its bounding box. The minimum x and y values were subtracted from the maximum x and y values in order to find the center co-ordinates of each blob. If there were unwanted pixels left over after the erosion of the mask, these would be classified as blobs too. In order to prevent this, the largest two blobs were used if there were more than two identified. The bounding box of each blob was drawn onto the frame by using the "rectangle" function in OpenCV. This bounding box was used to ensure that the programs estimation of the LED position and the actual LED position were the same.

Finding the Front LED

The front of the helicopter needed to be identified in order to find the orientation. Since the front LED on the helicopter was bright red and the back LED was white, the colour information could be used to differentiate them. It should be noted that the white LED came up quite blue in the frame. A loop was created which cycled through each pixel of each blob, and extracted the RGB values of that pixel. If the red value of the pixel was very high, a "red" count was increased and if the blue value of the pixel was very high, a "blue" count was increased. The front LED was identified as having more red pixels and less blue pixels than the back LED, or in the case of only one blob, the front LED was identified as having more red pixels and less blue pixels than a certain threshold. This threshold was obtained through experimentation.

Determining the Orientation

If there was only one blob found, then it was assumed that the helicopter must have been facing either North or South (North being the front of the helicopter facing the camera, South being the tail of the helicopter facing the camera). In order to determine which of these orientations was true, the check was performed to see if the blob was the front

LED or the back LED. If it was the front LED, it was assumed that the orientation of the helicopter was North. If it was the back LED, it was assumed that the orientation of the helicopter was South.

If there were two blobs found, then the helicopter could be facing East, West, or somewhere in-between. In order to find this orientation, the distance between the front and the back LEDs was calculated. Since this LED distance varied depending on the distance of the helicopter from the Kinect, it was necessary to conduct an experiment which measured the LED distance when the helicopter was placed at different distances from the Kinect. The results from this experiment are shown in Chapter 5 and were used to convert the different LED distances into orientation values which could be compared. These orientation values were obtained by multiplying the LED distance by the helicopter's distance from the camera, and dividing by 10000 to make them smaller and more manageable. The orientations East and West resulted in the maximum LED distance value and therefore the maximum orientation value. Since the distance was exactly the same if it were facing East or West, the front LED was identified in order to differentiate the two orientations. Values less than the maximum orientation value indicated that the helicopter was in more of a diagonal orientation such as North East or South West. The orientations North, South, East, and West were sufficient for the purposes of this system and therefore the orientation values for the other orientations were not calculated.

4.2.2 Combined CamShift and Thresholding

Due to the limitations associated with thresholding, which are discussed in Chapter 5, an additional method was needed to increase the robustness of the system. CamShift was combined with thresholding in order to limit the LED search space to a smaller area. Instead of thresholding the entire frame, the system only used the area in which the helicopter was located. The CamShift method that was used was based on the sample code provided with the OpenCV library.

OpenCV CamShift Demonstration Algorithm

The CamShift demonstration first requires the user to select the area which contains the object to be tracked. Once this area is established, the image is converted to the HSV colour space and the hue values of each pixel are extracted and put into their own matrix. The colour histogram of this new matrix is then calculated by using the OpenCV "calcHist" function.

The colour histogram is used, along with the hue matrix, to calculate the back projection of the incoming image frames. The OpenCV function "calcBackProject" is used which returns a grey-scale image with white areas meaning a very close match to the colour histogram and darker areas meaning that they are not a close match. In the case of the helicopter, its position in the image was very white in the back projection.

The next part of the OpenCV CamShift demonstration is responsible for resizing the search window, which is the area within the image that contains the highest matching area to the colour histogram. For the first image, the original box selected by the user is used but after that, the OpenCV "CamShift" function is used. The "CamShift" function uses the back projection and the previous search window to calculate the dimensions of the new search window. Once the new window is returned, an ellipse is drawn on to the image to show the estimation of the position of the object being tracked.

Altered CamShift

The OpenCV demo only tracked the area of the image that was the closest match to the colour histogram. This meant that the search window would only contain the front LED, as the front body of the helicopter was the area that contained the closest match to the colour histogram. The back LED would not fall within this area and consequently, it would not fall within the search window. This search window could therefore not be used as a search window to optimize the original tracking method. The OpenCV demo could be altered, however, to include the back LED in the search window. This was done by calculating the center co-ordinates of the new search window produced by the CamShift function. These co-ordinates were used to create a new search window which included the entire helicopter.

The window's dimensions were dynamic, using the distance of the helicopter from the Kinect to determine how large or small the search window needed to be. The search window's dimensions were calculated by placing the helicopter at a certain distance from the camera, facing either East or West. The distance between the two LEDs was calculated and multiplied by two. This value was used as the width of the search window as it ensured that the window would contain the helicopter at any orientation. The height of the window was determined by experimentation, ensuring that the resulting window

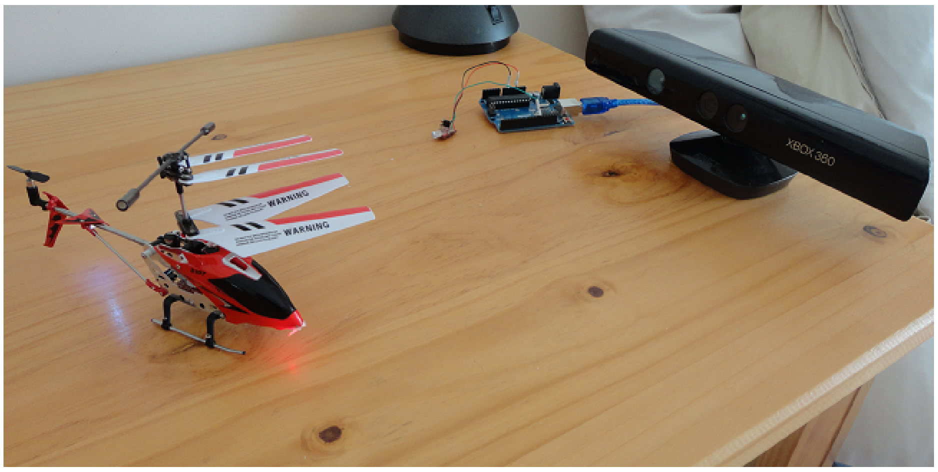

Figure 4.10: Setup of the auto-pilot system.

contained both LEDs. The helicopter's depth value from the Kinect depth sensor was calculated in order to determine how far away it was from the camera. This information was then used to calculate dynamic dimensions for the search window. The calculated dimensions were scaled depending on the distance of the helicopter from the camera.

4.3 Combined Tracking and Control

The tracking and control phases of the project were combined in order to control the helicopter autonomously. Figure 4.10 shows the finished system setup. The feedback control loop discussed in Chapter 3 was implemented where the helicopter was sent infrared signals to make it move in a certain way. These movements were tracked by the Kinect using the tracking algorithms already implemented. The feedback from these movements, in this case the co-ordinates of the helicopter, was extracted and analysed. Any noise produced in the feedback was counteracted by using erosion to get rid of unwanted blobs in the mask. The error was calculated by the difference between the desired co-ordinates and the helicopter's actual co-ordinates from the feedback. The error was corrected by adjusting the outgoing infrared signals being transmitted by the infrared LED. The desired position depended on the movement being performed by the helicopter. The different tests performed are discussed in Chapter 5.

4.4 Summary

The system was implemented in three separate phases which were control, tracking and combined tracking and control. The outcome of the control phase was the ability to control the helicopter using a computer and Arduino with an infrared LED. This phase aimed to reproduce the functionality of the original remote that came with the helicopter. The outcome of the tracking phase was the ability to track the helicopter using combined thresholding and CamShift. The helicopter's co-ordinates and basic orientation were extracted. The outcome of the last phase of the implementation was the ability to autonomously control the helicopter using combined tracking and control.

Chapter 5

Testing and Analysis

In order to test the auto-pilot program's functionality, a series of tests were implemented. The results of these tests showed the capabilities and limitations of the auto-pilot system. Each test was performed in the same environment which had quite bright artificial lighting and no exposure to sunlight. In addition to these tests, the results of the tracking methods implemented and the Kinect Depth Map test are discussed.

5.1 Accuracy of Kinect Depth Map

The depth map produced by the Kinect camera was used to find the distance of the helicopter from the Kinect, therefore the measurements needed to be accurate. To find the accuracy of the depth values given by the Kinect's depth sensor, an experiment was performed. A brightly coloured piece of paper was attached to an A4 book to make it easier to find the object's position. A measuring tape was placed on the floor, starting at the Kinect camera. The book was placed at certain distances on the measuring tape so that the actual distance could be compared to the distance produced by the Kinect depth map. A simple thresholding function was used which coloured pixels white if they matched the colour of the paper on the book, but coloured pixels black if the colour was different. This produced a mask showing the position of the book. The depth map produced by the Kinect was then searched and the depth value was recorded for every pixel that was coloured white in the mask. Figure 5.1 shows the setup of the experiment.

The raw depth values produced by the Kinect needed to be converted into meters, as they were recorded by default as 11-bit disparity values. The values were converted by using

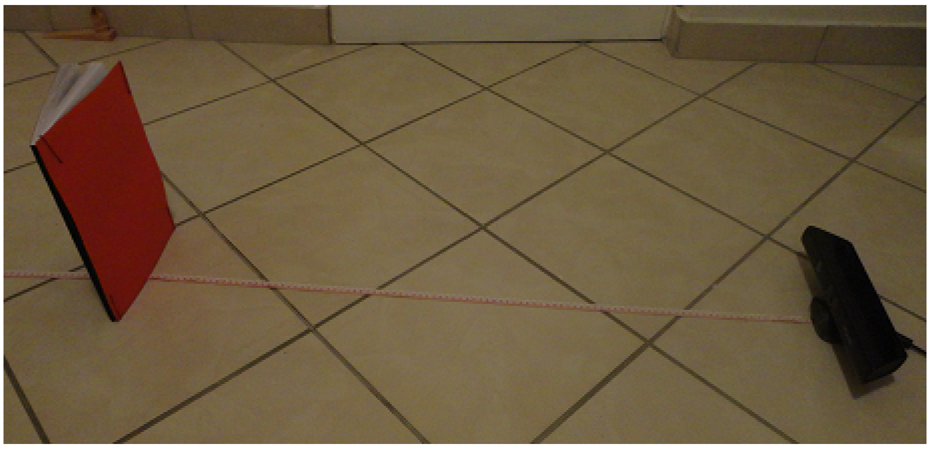

Figure 5.1: Setup of experiment to test the accuracy of the Kinect depth map using an A4 book.

the following formula, which was produced by Dr Stéphane Magnenat[1].

$$distance = 0.1236 * \tan(rawDisparity/2842.5 + 1.1863)$$

In addition to testing the depth values for the book, it was also necessary to test the accuracy of the depth values for the helicopter itself. This additional test needed to be performed because the helicopter is very small and its shape changed at different orientations.

5.1.1 Outcome

The book was placed at 800mm, 1000mm, 1200mm, 1500mm, and 2000mm. For each placement of the book, the distance produced by the Kinect was recorded and compared. The results can be seen in Table 5.1.

Table 5.1: Kinect Depth Map Accuracy Results when using an A4 book

Actual Distance (mm)	Computed Distance (mm)	Error (mm)
800	823.53	23.53
1000	1019.61	19.61
1200	1215.69	15.69
1500	1529.41	29.41
2000	2039.22	39.22

[1]Dr Stéphane Magnenat, http://stephane.magnenat.net/

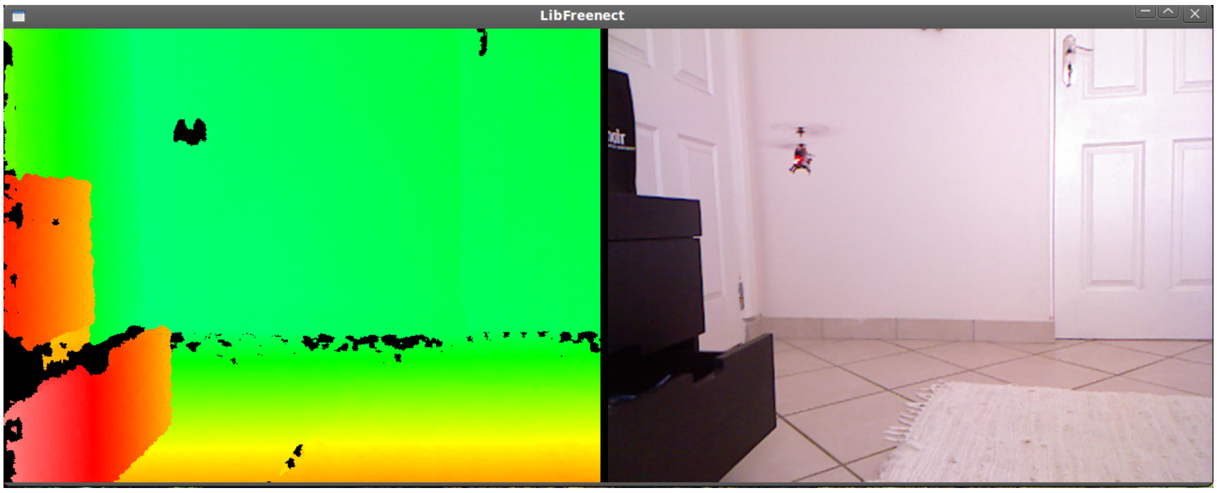

Figure 5.2: Kinect RGB camera and depth sensor output showing that no depth value could be assigned to the helicopter.

The results show that the Kinect depth map produces very accurate results and its maximum error is 39mm. The depth values produced when using the helicopter instead of the book, were just as accurate but only when the helicopter was facing East or West. If the helicopter was facing North or South, its shape became too small and no depth value was produced for it. Figure 5.2 shows the helicopter facing directly forward and how the depth map only shows black where the helicopter is situated. This indicates that no depth data could be found for that area.

This limitation was overcome by attaching a rectangular piece of card onto the front of the helicopter in order to increase its surface area. This addition did not affect the flying of the helicopter in any way. This solution worked very well and made the helicopter large enough to be assigned a depth value. This meant that a depth value could be produced for the helicopter at any orientation. Figure 5.3 shows that the helicopter is given a depth value when facing North when the piece of card is attached to it.

5.2 Determining the Orientation Results

The results for the experiment to find the orientation of the helicopter can be seen in Table 5.2. The Camera distance refers to the distance of the helicopter from the camera and the LED distance refers to the distance between the front and back LEDs on the helicopter. The orientation value refers to the multiplication of the previous two values divided by 10000. The orientation value takes into account the fact that the distance between the two

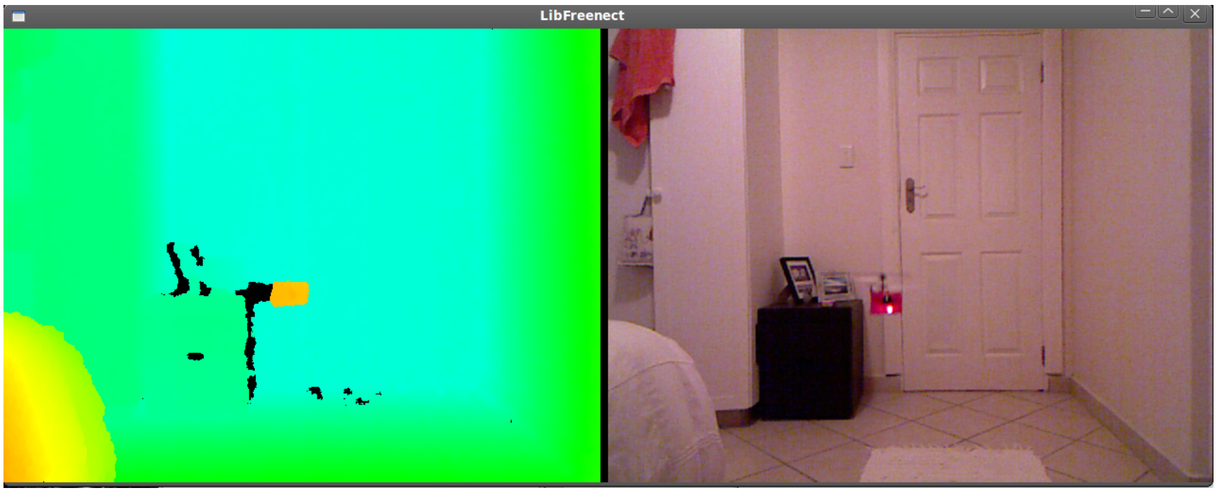

Figure 5.3: Kinect RGB camera and depth sensor output showing that a depth value was assigned to the helicopter.

LEDs varies depending on how far away the helicopter is from the camera. Therefore, the LED distance is combined with the camera distance in order to create an orientation value that stays constant when the helicopter is in a certain orientation, at different distances from the camera.

Table 5.2: Helicopter Orientation Results

Camera distance (mm) (CD)	LED distance (pixels) (LD)	Orientation Value (CD * LD)/10000
784	85	6.664
901	77	6.9377
980	67	6.566
1098	62	6.8076
1215	57	6.9255
1294	53	6.8582

The helicopter was facing East for the duration of the experiment. It was placed at a distance of 800mm, 900mm, 1000mm, 1100mm, 1200mm and 1300mm from the camera. The distance between the two LEDs was recorded for each of these distances and the orientation value was calculated. The results in Table 5.2 show that if the orientation value was between 6.5 and 7.0, it could be assumed that the helicopter was facing East or West, depending on the position of the front LED.

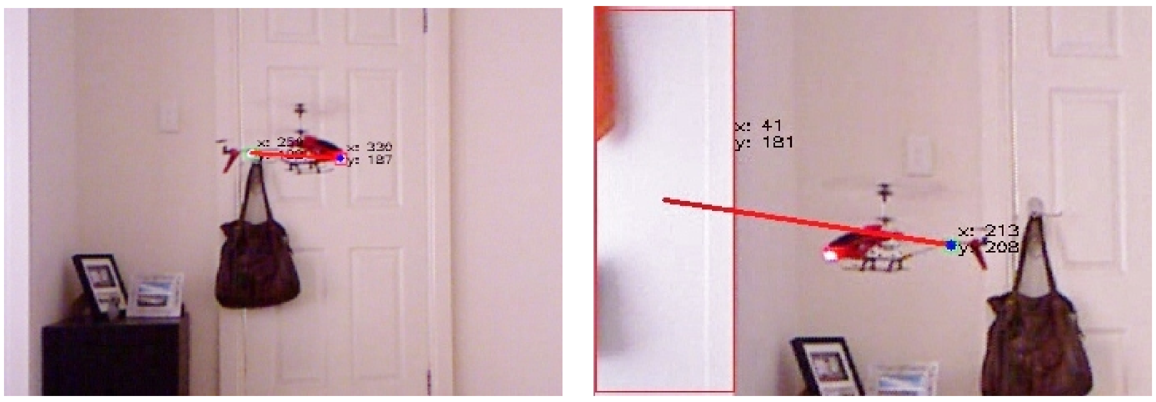

Figure 5.4: (a) The results of thresholding in suitable lighting conditions (b) The results of thresholding in bright lighting causing bright surfaces in the background.

5.3 Thresholding Results

In an ideal environment, where the LEDs on the helicopter were the brightest areas in the scene, the method of thresholding to find the LEDs in each image worked very well. Since thresholding is a very simple method, it requires a minimal amount of processing. The system was capable of processing approximately 54 frames per second (FPS) on a computer with an Intel i7 2.93GHz processor and 8GB of RAM. The Kinect can only produce a maximum of 30FPS, therefore the program could actually outperform the hardware. Figure 5.4(a) shows the results of thresholding in suitable lighting conditions. The helicopter's co-ordinates were extracted very accurately. However, in most environments there were some bright surfaces or reflections caused by the lighting in the room. This meant that the thresholding allowed the bright surfaces, other than the LEDs, to show up in the mask and were therefore labeled as blobs. This caused the method to fail because it could not tell the difference between the LED blobs and the background blobs. Figure 5.4(b) shows the results of thresholding in bright lighting which caused the method to fail. This limitation was partially overcome by making the threshold value dynamic instead of static. This solution worked well in situations where the background surfaces had a lower brightness value than the LEDs, but the method still failed in situations where the background surfaces had a higher brightness value than the LEDs.

5.4 Combined Thresholding and CamShift Results

The problems associated with pure thresholding were overcome by combining CamShift with thresholding. The search window was limited to the area in which the helicopter was

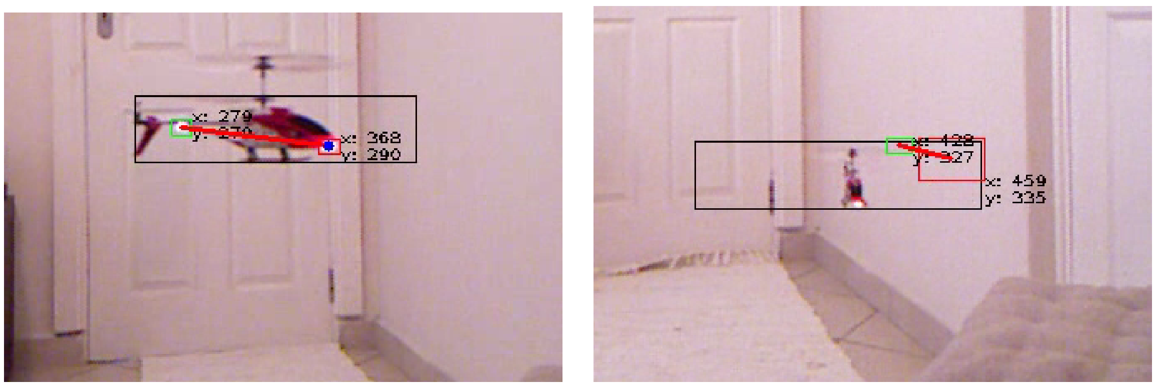

Figure 5.5: (a) The results of combining thresholding and CamShift for tracking the helicopter (b) A situation in which the combination of CamShift and thresholding still failed.

located. This meant that the thresholding was only performed in this area and therefore only the LEDs were included in the mask. Since CamShift requires a significantly larger amount of processing than thresholding, the performance of the system was decreased. This decrease did not have a negative impact on the system, however, as the frame rate was decreased from 54FPS to approximately 35FPS on the same computer and this is still larger than the Kinect's maximum of 30FPS. If 35 frames were processed per second, one frame was processed in approximately 28ms. This time was used in Chapter 4 when calculating the delay of data sent to the Arduino. Figure 5.5(a) shows the results of combining thresholding with CamShift. The black box indicates the search window produced by CamShift. This method failed, however, when the helicopter moved into one of the bright areas in the background, as the bright areas would also be included in the search window and consequently the mask. Figure 5.5(b) shows a situation in which the combined CamShift and thresholding failed, as the helicopter moved in front of a bright surface.

The main advantage of using the CamShift and thresholding methods together was that each of them effectively cancelled out the weaknesses of the other. The thresholding method could not be used on its own because even though it could accurately track the position of each LED, it could not function in bright lighting. CamShift could not be used on its own because even though it worked very well in bright lighting, it could only the track the whole helicopter and not the individual LEDs. By combining the two, both their limitations were overcome and the helicopter could be tracked effectively.

5.5 Hovering

The first test involved keeping the helicopter hovering in the center of the screen in order to test the speed of the control loop. This test was performed by manually flying the helicopter upwards to the desired hovering point. This point was recorded and used as a reference. The more the helicopter moved above the reference point, the more the throttle was decreased. The more the helicopter fell below the reference point, the more the throttle was increased.

5.5.1 Outcome

The helicopter's throttle was decreased or increased as it fell above or below the reference point, but the helicopter's reactions were slighty delayed. The system changed the control values as soon as it detected a difference between the reference point and the actual position, but due to the inertia of the helicopter there was a slight delay between the changes in movement of the rotors. This caused the helicopter to oscillate above and below the reference point. This is also referred to as pilot induced oscillation, where overcorrection occurs due to delays in the movement of the aircraft. These delays are caused by the rotors on the helicopter having a fixed pitch, which was discussed in Chapter 3. The helicopter also tended to drift around the room, until it moved out of the frame. This drifting could either be due to gusts of wind or an imbalance in the helicopter. This drifting was partially overcome by moving the system into a larger area in order to prevent the gusts. The initial manual throttle increase needed to move the helicopter up into the frame had to be changed for every run due to differences in the battery charge. If the battery was fully charged, the throttle had to be quite low, but if the battery had a low charge, the throttle had to be high. Battery life was not considered when deciding on the type of helicopter to use in the system. This battery life could be made longer by replacing the original 3.7V 150mAh battery with a 3.7V 240mAh battery. Another solution, used for this system, is to have more than one helicopter for testing.

5.6 Flying in a Square

The second test involved using orientation and depth in order to get the helicopter to fly in a horizontal square, at a constant altitude. The helicopter had to start at some altitude in the frame, facing North. It then had to turn right until it was facing East. If it was facing East, the helicopter had to fly forwards until it reached a certain distance. It then had to turn right until it was facing South. If it was facing South, the helicopter had to fly forwards until it reached a certain distance. Once it had reached the correct distance, the helicopter stopped flying forward and turned right until it was facing West. If it was facing West, the helicopter had to fly forwards until it reached a certain distance. It then had to turn right until it was facing North. If it was facing North, the helicopter had to fly forward until it reached a certain distance. Once it had reached the correct distance, the helicopter stopped flying forward.

5.6.1 Outcome

The turning of the helicopter and the checking of its orientation worked well. Each frame was checked to see if the helicopter was at the right orientation. If it was not at the right orientation, the helicopter carried on turning right but if it was at the right orientation, it stopped turning and began with the next step of the movement. The flying forward worked well when the helicopter was facing East and West as the x co-ordinate of the helicopter was extracted in each frame and as soon as it was equal to the given distance, the yaw of the helicopter was increased in order to stop it from moving forward. The flying forward did not work originally, when the helicopter was facing North or South because the distance of the helicopter from the camera had to be found and this could not be done when the helicopter was at those orientations, as discussed earlier in this chapter. However, the previous solution to this problem was used again, where a piece of card was attached onto the helicopter making it larger in size, in order to be assigned a depth value. This problem could have also been resolved by turning the helicopter to face East every few frames so that its distance could be checked.

5.7 Navigating to a Brightly Coloured Target

The third test involved navigating the helicopter to a stationary target in order to test the accuracy of the movement of the helicopter. The target was a bright blue block situated at some random position in the frame. Its depth and coordinates had to be extracted by using a simple colour thresholding function to extract the target and identify its position. The helicopter had to start at some height in the frame facing North. It had to use its depth and position and the targets depth and position in order to calculate how and where it should move.

5.7.1 Outcome

The system calculated the helicopter's initial co-ordinates and distance from the camera, as well as the object's co-ordinates and distance from the camera. The distances of the helicopter and the object were compared and if the object was in front of the helicopter, the helicopter turned to face North and moved until it reached the distance of the object. If the object was behind the helicopter, the helicopter turned to face South and moved until it reached the distance of the object. The x co-ordinates of both the helicopter and the object were compared and if the object was to the West of the helicopter, the helicopter turned to face West and moved forward until it reached the object's x co-ordinate. If the object was to the East of the helicopter, the helicopter turned to face East and moved forward until it reached the object's x co-ordinate. The y co-ordinates of both the helicopter and the object were compared and if the object was below the helicopter, the helicopter's throttle was decreased until it reached the object's height. If the object was above the helicopter, the helicopter's throttle was increased until it reached the object's height. The drifting of the helicopter meant that the results were varied and it would sometimes move off target or out of the frame. The delay in the rotors of the helicopter resulted in it sometimes moving too far upwards or downwards.

5.8 Summary

The results from the tests performed on the system were positive, apart from a few limitations and challenges identified. The fact that the helicopter was not able to be tracked successfully when flying over bright areas was a minor limitation of the object tracking algorithms used. Limiting the environment to a certain extent is a very common solution for object tracking and therefore this problem could be overcome by limiting the system to environments where bright surfaces are not located near the center of the background. The small delay in the changes of the helicopter's movements caused the system to overcorrect, and therefore its movements were not completely accurate. Since the system was not expected to perform highly accurate and advanced movements, this delay did not have a significant impact on the results. The main challenge faced by the hardware was the varying performance of the helicopter, as a result of the rapidly decreasing battery charge. However, this did not have a negative impact on the results, it just caused them to vary for each test.

Chapter 6

Conclusion

The system that was created simulates an auto-pilot system for a mini remote controlled helicopter. It enables the helicopter to perform certain movements by tracking and controlling it autonomously. The object tracking functions that were used were very efficient and resulted in the system being fast enough to maintain control. A combination of methods, namely thresholding and CamShift, were used simultaneously for object tracking in order to increase the robustness of the system. Although there were certain limitations and challenges associated with the hardware and algorithms used, the system was successful and was able to fulfil the requirements and objectives stated in Chapter 1.

6.1 Future Work

This system could be recreated using a larger air vehicle, such as a quadrocopter, which is more expensive than the helicopter being used in this system, but much more stable making it easier to control. A larger air vehicle would also mean a larger battery could be used which would simplify testing and make the system more robust and predictable. The system could also be improved in the future by using more than one air vehicle which could be networked to perform synchronised movements.

The role of automation in the system could be changed by using on-board intelligence instead of having only external intelligence. The Kinect could be replaced with a smaller camera which could be attached to the helicopter and the controlling unit would have to be connected to the helicopter. Having the camera on-board would result in the system being better at navigating around obstacles, which would mean the helicopter could follow a given path, avoiding any objects obstructing that path.

Bibliography

[1] Aguilar-Ponce, R. *Automated object detection and tracking based on clustered sensor networks*. PhD thesis, University of Louisiana, 2007. AAI3294839.

[2] Allen, J. G., Xu, R. Y. D., and Jin, J. S. Object tracking using camshift algorithm and multiple quantized feature spaces. In *Proceedings of the Pan-Sydney area workshop on Visual information processing* (Darlinghurst, Australia, Australia, 2004), VIP '05, Australian Computer Society, Inc., pp. 3–7.

[3] Altug, E., Ostrowski, J. P., and Taylor, C. J. Quadrotor control using dual camera visual feedback. In *International Conference on Robotics & Automation* (2003), IEEE, pp. 4294–4299.

[4] Amidi, O., Mesaki, Y., and Kanade, T. Research on an autonomous vision-guided helicopter, 1993.

[5] Andersen, M., Jensen, T., Lisouski, P., Mortensen, A., Hansen, M., Gregersen, T., and Ahrendt, P. Kinect depth sensor evaluation for computer vision applications. Tech. rep., Aarhus University, 2012.

[6] Belongie, S., Malik, J., and Puzicha, J. Shape matching and object recognition using shape contexts. *IEEE Transaction on Pattern Analysis and Machine Intelligence 24*, 4 (Apr. 2002), 509–522.

[7] Benezeth, Y., Jodoin, P., Emile, B., Laurent, H., and Rosenberger, C. Review and evaluation of commonly-implemented background subtraction algorithms. In *Pattern Recognition* (2008), pp. 1–4.

[8] Castillo, P., Lozano, R., and Dzul, A. E. *Modeling and control of mini-flying machines*. Advances in industrial control. Springer, 2005.

[9] Cheng, Y. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell. 17*, 8 (Aug. 1995), 790–799.

[10] Comaniciu, D., Ramesh, V., and Meer, P. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25* (2003), 564–577.

[11] Doyle, J., Francis, B., and Tannenbaum, A. *Feedback Control Theory.* Macmillan Publishing Co., 1990.

[12] Emami, E., and Fathy, M. Object tracking using improved camshift algorithm combined with motion segmentation. In *IEEE Machine Vision and Image Processing* (2011), pp. 1–4.

[13] Fan, J., Yau, D., Elmagarmid, A., and Aref, W. Automatic image segmentation by integrating color-edge extraction and seeded region growing. *IEEE Transaction on Image Processing 10* (2001), 1454–1466.

[14] Fieguth, P., and Terzopoulos, D. Color-based tracking of heads and other mobile objects at video frame rates. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* (1997), pp. 21–27.

[15] Fukunaga, K., and Hostetler, L. D. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory 21*, 1 (1975), 32–40.

[16] Gabriel, P., Hayet, J. B., Piater, J., and Verly, J. Object tracking using color interest points. In *IEEE Proceedings of International Conference on Advanced Video and Signal based Surveillance* (2005), pp. 159–164.

[17] Gavrilets, V., Frazzoli, E., Mettler, B., Piedmonte, M., and Feron, E. Aggressive maneuvering of small autonomous helicopters: A human-centered approach. *The International Journal of Robotics Research 20* (2001), 795–808.

[18] Gonzalez, R., and Woods, R. *Digital Image Processing.* Pearson/Prentice Hall, 2002.

[19] Heath, M., Sarkar, S., Sanocki, T., and Bowyer, K. Comparison of edge detectors: A methodology and initial study. In *Computer Vision and Image Understanding* (1996), IEEE Computer Society Press, pp. 38–54.

[20] Hidayatullah, P. Object tracking: State of the art and camshift improvement using multi-dominant colors tracking. Master's thesis, University of Jean Monnet Saint-Etienne,, 2010.

[21] Intel Corporation. *Open Source Computer Vision Library Reference Manual,*, 2001.

[22] Jiang, X., and Bunke, H. Edge detection in range images based on scan line approximation. *Computer Vision and Image Understanding 73* (1999), 183–199.

[23] McGillivary, P., Sousa, J., Martins, R., Rajan, K., and Leroy, F. Integrating autonomous underwater vessels, surface vessels and aircraft as persistent surveillance components of ocean observing studies. In *IEEE Autonomous Underwater Vehicles* (Southampton, UK, 2012).

[24] Moeslund, T. B., and Granum, E. A survey of computer vision-based human motion capture. *Computer Vision Image Understanding 81*, 3 (Mar. 2001), 231–268.

[25] Ning, J., Zhang, L., Zhang, D., and Wu, C. Robust object tracking using joint color-texture histogram. *IJPRAI* (2009), 1245–1263.

[26] Nummiaro, K., Koller-Meier, E., and Gool, L. V. Color features for tracking non-rigid objects. *Special Issue on Visual Surveillance, Chinese Journal of Automation 29* (2003), 345–355.

[27] OpenCV. Basic thresholding operations, 2012.

[28] Pavlidis, T., and Liow, Y. T. Integrating region growing and edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence 12*, 3 (Mar. 1990), 225–233.

[29] Roh, M.-C., Kim, T.-Y., Park, J., and Lee, S.-W. Accurate object contour tracking based on boundary edge selection. *Pattern Recognition 40* (March 2007), 931–943.

[30] Rosenhahn, B., Kersting, U., Andrew, S., Brox, T., Klette, R., and Seidel, H.-P. A silhouette based human motion tracking system. Tech. rep., University of Auckland, 2005.

[31] Schmid, C. Introduction into system control, 2005.

[32] Sneep, P., and Richner, J. Tracking system and communication interface for miniature rc helicopters. Hello, 2011.

[33] Solomon, C. J., and Breckon, T. P. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab.* Wiley-Blackwell, 2010. ISBN-13: 978-0470844731.

[34] Sommerfeld, J. Image processing and object tracking from single camera. Master's thesis, Royal Institute of Technology (KTH), 2006.

[35] Sonka, M., Hlavac, V., and Boyle, R. *Image Processing, Analysis and Machine Vision*, 2 ed. Brooks/Cole, 1999.

[36] Stauffer, C., and Grimson, W. E. L. Adaptive background mixture models for real-time tracking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (1999), vol. 2, IEEE Computer Society, pp. 246–252.

[37] Sugandi, B., Kim, H., Tan, J. K., and Ishikawa, S. *Object Tracking.* InTech, 2011, ch. 1, pp. 3–22.

[38] Tarel, J.-P., and Hautiere, N. Fast visibility restoration from a single color or gray level image. In *Computer Vision, IEEE 12th International Conference* (2009), pp. 2201–2208.

[39] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Gale, P. F. J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., and Stang, P. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics 23* (2006), 661–692.

[40] Veenman, C. J., Hendriks, E. A., Veenman, C. J., Hendriks, E. A., and Reinders, M. J. A fast and robust point tracking algorithm. In *In IEEE International Conference on Image Processing, volume III* (1998), pp. 653–657.

[41] Xu, R. Y. D., Allen, J. G., and Jin, J. S. Robust real-time tracking of non-rigid objects. In *Proceedings of the Pan-Sydney area workshop on Visual information processing* (Darlinghurst, Australia, Australia, 2004), VIP '05, Australian Computer Society, Inc., pp. 95–98.

[42] Yilmaz, A., Javed, O., and Shah, M. Object tracking: A survey. *ACM Computing Surveys (CSUR) 38* (2006), 45.

[43] Yin, F., Makris, D., and Velastin, S. Performance evaluation of object tracking algorithms, 2007.