USB Camera Pedestrian Counting

A dissertation submitted by

Jeremy Bruce Duncan

in fulfilment of the requirements of

Courses ENG4111 and 4112 Research Project

towards the degree of

Bachelor of Engineering (Elec)

Submitted: 31\textsuperscript{st} of October 2010
LIMITATIONS OF USE

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof Frank Bullen
Dean
Faculty of Engineering and Surveying
Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Jeremy Duncan

Student Number: 0050012967

___________________________________

signature

___________________________________

date
# Table of Contents

1. ABSTRACT ............................................................................................................ 8
2. ACKNOWLEDGEMENTS .................................................................................... 9
3. LIST OF FIGURES ............................................................................................ 10
4. INTRODUCTION .............................................................................................. 12
5. PROJECT OBJECTIVES .................................................................................. 13
6. BACKGROUND ................................................................................................. 14
   6.1 Literature Review ...................................................................................... 14
      6.1.1 Aim ...................................................................................................... 14
      6.1.2 Reviews .............................................................................................. 14
   6.2 Summary of People Tracking Methods ...................................................... 23
      6.2.1 People Tracking Algorithms .............................................................. 23
      6.2.2 A Linear Prediction Tracker .............................................................. 23
      6.2.3 Head Detectors .................................................................................. 24
      6.2.4 The Leeds People Tracker ................................................................. 24
      6.2.5 The Reading People Tracker ............................................................. 25
   6.3 Motion Detection ......................................................................................... 27
      6.3.1 Frame Difference Method ................................................................. 27
      6.3.2 Average Filter Method ...................................................................... 27
      6.3.3 Median Filter Method ....................................................................... 28
      6.3.4 Running Average Method .................................................................. 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.5</td>
<td>Kalman Filter</td>
<td>29</td>
</tr>
<tr>
<td>6.3.6</td>
<td>Other Filters</td>
<td>29</td>
</tr>
<tr>
<td>6.4</td>
<td>Tracking Methods</td>
<td>30</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Active Shape Tracking</td>
<td>30</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Region Tracking</td>
<td>30</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Mean Shift Tracking</td>
<td>31</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Feature Tracking</td>
<td>31</td>
</tr>
<tr>
<td>6.5</td>
<td>Research Summary</td>
<td>33</td>
</tr>
<tr>
<td>6.6</td>
<td>Initial System Design</td>
<td>35</td>
</tr>
<tr>
<td>6.7</td>
<td>Programming Language Selection</td>
<td>38</td>
</tr>
<tr>
<td>6.8</td>
<td>Project Resources</td>
<td>42</td>
</tr>
<tr>
<td>6.9</td>
<td>Basic Terminology</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>DESIGN AND BUILD</td>
<td>43</td>
</tr>
<tr>
<td>7.1</td>
<td>Section Overview</td>
<td>43</td>
</tr>
<tr>
<td>7.2</td>
<td>Design and Build Method</td>
<td>44</td>
</tr>
<tr>
<td>7.3</td>
<td>Image Acquisition</td>
<td>44</td>
</tr>
<tr>
<td>7.4</td>
<td>Greyscale Conversion</td>
<td>48</td>
</tr>
<tr>
<td>7.5</td>
<td>The Background Image</td>
<td>50</td>
</tr>
<tr>
<td>7.5.1</td>
<td>The Ghost Filter Variable</td>
<td>53</td>
</tr>
<tr>
<td>7.6</td>
<td>The Difference Image</td>
<td>53</td>
</tr>
<tr>
<td>7.7</td>
<td>The Pixelator</td>
<td>57</td>
</tr>
</tbody>
</table>
USB Camera Pedestrian Counting

7.7.1 Pixelator Code ................................................................................. 60

7.8 Software Engineering ........................................................................... 62

7.8.1 Multithreading Trials ...................................................................... 62

7.8.2 Structured Programming ................................................................. 63

7.9 Object Growing .................................................................................... 63

7.9.1 Region Growing by Seeding ............................................................ 65

7.9.2 Line by Line Region Growing .......................................................... 68

7.9.3 Line Blob Formation ...................................................................... 69

7.9.4 Blob Merging .................................................................................. 71

7.9.5 Region Formation ........................................................................... 72

7.9.6 Region Growing Results ................................................................. 78

7.10 Basic Object Tracking ........................................................................ 82

7.11 Final Code .......................................................................................... 88

7.12 Practical Deployment .......................................................................... 128

7.13 Linux Deployment ............................................................................... 130

7.14 Other Applications ............................................................................ 131

8 CONCLUSIONS .......................................................................................... 132

8.1 Result Project Achievements ............................................................... 132

8.2 Recommendations For Future Work .................................................. 137

8.2.1 Difference Image Improvements ...................................................... 137

8.2.2 Image Size Reduction ..................................................................... 138
1 ABSTRACT

The aim of this project was to implement a pedestrian counting system using a PC and USB Camera as the primary hardware. The software developed will not be ready for complete deployment due to time limitations and requires further development before it is reliable and accurate enough to be used for pedestrian counting. However, the object motion detector has been fully developed and is ready to be incorporated into future projects and currently runs at 26 frames per second.

The current program captures frames in real time from a USB camera. A motion image is created using an approximate median filter. A motion image is then generated using differencing. Moving objects are clustered using a region growing algorithm. These motion objects are then displayed on screen. Tracking at this stage consists of simple size and position matching combined with aging of the objects to increment a pedestrian counter.

Further development of the project will involve enhanced tracking methods such as region splitting, active model fitting, velocity and position estimates using predictor correctors and shadow removal. Difference image averaging should be applied to improve the results and robustness of the motion detector which is currently noisy. Other improvements would be the transition of the program to a C language to improve speed along with multithreading, greater camera control and enhanced statistics reporting.
2 ACKNOWLEDGEMENTS

Many thanks go to John Billingsley of the University of Southern Queensland who has assisted with the project and acted as the author’s supervisor.

Also many thanks for the articulate documentation provided by Nils Sibel, developer of the Reading People Tracker, whose well documented work guided the approach taken by the author.
3 LIST OF FIGURES

FIGURE 1 - READING TRACKER MODULE DESCRIPTION 25
FIGURE 2 - READING PEOPLE TRACKER ALGORITHM 26
FIGURE 3 - ALGORITHM OVERVIEW 35
FIGURE 4 - APPROXIMATE MEDIAN FILTER 35
FIGURE 5 – INITIAL FEATURE TRACKER DESIGN 37
FIGURE 6 – OSNEWS LANGUAGE PERFORMANCE COMPARISON 40
FIGURE 7 – PIXEL TO ARRAY LAYOUT 46
FIGURE 8 - 1ST WEBCAM CAPTURE 47
FIGURE 9 - DIFFERENCE IMAGE EXAMPLE 1 54
FIGURE 10 - DIFFERENCE IMAGE EXAMPLE 2 55
FIGURE 11 - DIFFERENCE IMAGE EXAMPLE 3 56
FIGURE 12 - SEGREGATION IN THE DIFFERENCE IMAGE 57
FIGURE 13 - PIXELATOR RESULTS 58
FIGURE 14 - GROWING OBJECTS FROM A DIFFERENCE IMAGE 64
FIGURE 15 - GROSS REGION SEEDING 65
FIGURE 16 - COMPREHENSIVE SEED GROWING 66
FIGURE 17 - PROPOSED SEEDING ALGORITHM 66
FIGURE 18 - ADVANCED REGION GROWING ALGORITHM 68
FIGURE 19 - SIMPLIFIED METHOD FOR GROWING REGIONS 69
FIGURE 20 - LINE BLOBS EXAMPLE 70
FIGURE 21 - ROW SCANNER ALGORITHM 71
FIGURE 22 - LINE BLOB COLLISION DETECT 71
FIGURE 23 - LINE BLOB COLLISION LOGIC 72
FIGURE 24 - REGION GROWTH LOGIC TESTS 74
FIGURE 25 - REGION OVERLAP 75
FIGURE 26 - REGION COLLISION LOGIC 76
4 INTRODUCTION

The research project’s primary aim is to create a working pedestrian counter which uses a USB camera as its primary input sensing device. A camera would be mounted overlooking the pedestrian pathways and basic statistics such as quantity and frequency of pedestrians would be recorded.

People counting is currently performed manually, or using recorded video which is later played back and once again manually counted, and also by simple sensors (light curtains or pressure pads) which trigger when crossed or are pressed. The original intention was to produce a product for the Toowoomba City Council. The counting of pedestrians was to be used to evaluate if a pathway was to be added or to be upgraded.

Research shows that there are currently commercial systems available with the objective of people tracking. Part of the rationale for the project proposed by Sam Cubero of the USQ mechatronics department, the project originator, included the high software license costs associated with these systems. The software if successful could be made publically available on the internet. A low cost USB camera and laptop deployed in the right location could provide retail outlet owners, councils and others with a cheap method of counting pedestrians.
5 PROJECT OBJECTIVES

The following objectives for the project have been set.

1. Research and identify the most appropriate programming language for the project and develop a working knowledge of the chosen language.
2. Research current theories and algorithms used in the field of vision systems, shape and pattern recognition and object tracking.
3. Design and write the software.
4. Test the software and record the results.
5. If the written program is successful in a basic test environment, trial the system in more difficult conditions, identify flaws, and improve the program resiliency to changes in camera perspective and lighting.

As time permits:

6. Discuss system costs in terms of computer hardware and mounting enclosure required for practical installations.
7. Consider developing the system for linux to lower costs using a cross platform language.
8. Consider using the software for vehicular traffic and the changes to the software required.
9. Consider using the software for traffic light control enhancement.
10. Identify other applications for this type of system.
6 BACKGROUND

6.1 Literature Review

6.1.1 Aim

The aim of this literature review is to identify methods which can be easily implemented within the given timeframe and which will deliver a working person tracker.

6.1.2 Reviews


Relevance: High – Software and Required Algorithms Referenced and Code Available

Key Terms: Active Shape Model, Region/Blob Based Tracking, Principle Component Analysis, Motion Detector, Active Shape Tracker, Head Detector, Haritaoglu’s W4 System, Background Image, Pixel Difference.

Article Summary: The article directly relates to the objectives of this research project. The article describes the design and implementation of a people tracking software system. 4 different tracking methods are combined to improve the
resiliency of the software. The main modules are a Motion Detector, a Region Tracker, a Head Detector and an Active Shape Tracker and these modules exchange their results to improve the reliability. Low to medium intensity algorithms are used which is preferable to keep the system real-time. Source code is also available in C++. The tracker is said to be reliable in the presence of occlusion and low image quality. The “Reading People Tracker” developed is an extension of the “Leeds People Tracker”.

“Background Subtraction and Shadow Detection in Grayscale Video Sequences” (Jacques et al, 2005)

**Relevance:** Medium – algorithms employed are well documented and usable.

**Key Terms:** Medium Filter, Background Image, Shadow Removal

**Article Summary:** This article proposes a method of background subtraction which also detects and removes shadows. The researchers base their algorithm on the W4 system which is a median filter. The shadow filter can assist to overcome the issue of moving objects being connected by shadow. This filter will be used if more effective means cannot be found and also Shadow removal filter will also be used once again in the absence of more effective alternatives. There are still some issues remaining with the shadow detection proposed according to the author.
“A Neural Network for Image Background Detection” (Avent & Neal, 1995)

Relevance: Low

**Article Summary:** This article describes a method for background detection which relies on being to select the colour of the background hence the processing time dedicated to background detection can be significantly reduced. This method is not adaptive and hence is unsuitable for the project.

“A Moving Objects Detection Algorithm Based on Improved Background Subtraction”

Relevance: Medium

**Article Summary:** This article identifies some of the current methods of motion detection, namely the optical flow method, Consecutive Frame Subtraction and Background Subtraction. It identifies background subtraction as the most effective. Unfortunately due to the poor translation, the article is difficult to understand when it becomes more technical. This article will be explored more fully only if other motion detectors cannot be found. (Niu & Jiang, 2008)

“The Algorithm of Moving Human Body Detection Based On Region Background Modeling” (Fan & Li, 2009)

Relevance: High

**Article Summary:** The article describes a motion detector which shows high quality results and will adapt to changing environments. The algorithm is based on region background modelling. The complexity of the algorithm however is cause for
concern due to the time which will be required to implement it plus its high processing cost which results from this complexity. The steps in the algorithm are clear and show all formulas required. This article also describes some of the current methods of background detection and the relative strengths and weaknesses.

“Universal Serial Bus Device Class Definition for Video Devices Revision 1.1” (Intel Corp et al)

**Relevance:** Low

**Article Summary:** This article defines the USB Video Device standard and is primarily directed towards developers. This article was explored to determine what would be required in order to communicate with the USB camera and retrieve images. Further research into Visual Basic shows that the AviCap32.dll will meet the needs of this project.

“Teach Your Old Web Cam New Tricks: Use Video Captures in Your .NET Applications” (Wei Meng)

**Relevance:** Medium

**Article Summary:** This article demonstrates how to capture images from a USB device using the Basic language. It describes how to generate a form, use AVICap32.dll, and select the video source and then either capture a video sequence or a single image. A trial with the steps described in the article was described and an
image was successfully captured and saved as a BMP. Further research needs to be performed on how to take this BMP and store it in a matrix for manipulation.

“Tracking People” (Kim & Ranganath, 2002)

Relevance: Low-Medium

Article Summary: Colour based tracking is used in this system. Variable bin widths are used for storing the object histograms. Heuristics are used for issues such as occlusion and a person re-entering a scene. Details are few however and it would be difficult to extract any usable modules from the system.

“Automatic Counting Of Interacting People By Using A Single Uncalibrated Camera” (Velipasalar et al, 2006)

Relevance: Medium Low

Article Summary: This system relies on the camera mounting position to overcome occlusion issues. Fast blob tracking and the mean shift tracking algorithms are used. An entry and exit line must also be clearly available which is a valuable idea, but only if both entry and exit can occur on the same line. This system is not particularly adaptive.
“Tracking Multiple People for Video Surveillance” (Ali, Indupalli & Boufame)

**Relevance:** Medium

**Article Summary:** This system uses Background Subtraction and a Correlation based feature tracking object tracker. It categorises motion detectors as Frame Differencing Techniques, Background Subtraction and Optimal Flow. It categorises object detectors as Region-based tracking, Active-contour-based tracking, Feature-based tracking and Model-based tracking. To generate blobs, a seeding algorithm is implemented after a motion image has been generated. Exhaustive blob matching is used whereby a blob is checked against all existing blobs and a match is found. It opts for a feature based tracking system and tracks the features by using the Blob Histogram, Motion and Size. It then performs a correlation calculation between all blobs past and present with matches being made based on the highest correlation coefficient.

“Real-Time Tracking of Multiple People Using Continuous Detection” (Beymer & Konlige, 2000)

**Relevance:** Low

**Article Summary:** This tracker uses stereo inputs and hence will be unsuitable for the project.
“Robust techniques for background subtraction in Urban Traffic Video” (Cheung & Kamath)

**Relevance:** High

**Article Summary:** This article compares several background subtraction techniques. In summarises by saying that the Gaussian Mixture method offers the best results, however the Median filter offers similar results and is significantly simpler in construction. The memory consumption of the Median filter is of concern.

“A Kalman Filter Based Background Updating Algorithm Robust To Sharp Illumination Changes” (Segata et al)

**Relevance:** Medium

**Article Summary:** This algorithm uses a Kalman filter and tries to address the Kalmans filters inability to deal with global and sharp illumination changes. Methods to measure noise variance are discussed to deal with the issue of pixel saturation.


**Relevance:** Low-Medium

**Article Summary:** Backgrounds are first modelled using an empty scene. A large changing region is tracked and if the size is sufficient, a blob is built. 2D contour shape analysis Ids hands feet and head and a flesh like colour is applied. Other blob
areas are filled with cloth like colouring. The system can only cope with one person in the scene and does not adapt to variation in lighting.

“Tracking Of Pedestrians - Finding And Following Moving Pedestrians In A Video Sequence” (Siken, 2009)

**Relevance:** Medium

**Article Summary:** Contains some simple methods for object tracking such as geometric rules and colour tracking. These methods would be unsuitable for tracking multiple objects.

“A Mean-Shift Tracker: Implementations In C++ And Hume” (Wallace, 2005)

**Relevance:** Medium High

**Article Summary:** The article describes the means shift tracking system with a focus on implementation. The mean shift tracking theorem does not require the typical background subtraction method. A tracking box is created after which tracking of a region occurs. While theoretical details are sparse, implementation is well documented. The running speed of the system is 21.2 seconds for 150 frames at a resolution of 320*240 running on a Dual 933MHz machine. When referring to some of the sources within the article for theoretical background, the high majority of the theory had not been previously encountered.
“Mean-Shift” (Wikipedia, 2010)

**Relevance:** Medium-Low

**Article Summary:** This gives a brief introduction to the mean shift tracking algorithm.

“Accurate Real-Time Object Tracking With Linear Prediction Method” (Yeoh & Abu-Bakar, 2003)

**Relevance:** Medium-High

**Article Summary:** This describes a system capable of tracking a single object. It uses edge detection followed by a 2nd order linear predictor-corrector method. It claims to be more accurate than a Kalman type predictor however the tests appear limited.

“Rapid And Robust Human Detection And Tracking Based On Omega-Shape Features” (Li et al, 2009)

**Relevance:** Medium

**Article Summary:** This article uses 2 combined head and shoulder detectors, namely the Viola-Jones type classifier and a local histogram of oriented gradients (HOGs) feature based classifier. After detection a particle filter tracks the head/should combination. It is meant to be effective in the presence of partial
occlusion and crowded areas and shows a low computation time per detection and track. Details are sparse however regarding implementation.

6.2 Summary of People Tracking Methods

6.2.1 People Tracking Algorithms

Some of the available complete algorithms will now be explained to give an overview of how people tracking has been achieved by various researchers. This listing is far from exhaustive and is only presented to demonstrate some of the more common approaches encountered. It is possible that a hybrid algorithm may be developed from within the modules identified.

6.2.2 A Linear Prediction Tracker

This system uses an edge detection routine which involves an edge detection filter followed by a frame difference, followed by thresholding and flattening the result into a binary motion image. A centroid is fit to those edges using the histogram projection technique. A 2nd order linear predictor solved by the maximum entropy method is used for tracking centroids (Yeoh&Abu-Bakar, 2003).
6.2.3 Head Detectors

Some trackers focus on the upper part of the body to minimise issues with occlusion. Due to the omega like shape of the head and shoulders, and its nature to be generally at the top of a person like region it can be more easily described. These types of systems are sometimes referred to as Omega detectors. One system encountered using multiple head and shoulder detectors to increase initial detection within an entrance zone followed by a particle filter tracker (Li, 2009).

6.2.4 The Leeds People Tracker

Background subtraction is used to generate a motion image. The background is updated when pixels are shown to be decreasing or increasing in a regular fashion which attempts to avoid alternating changes and adapts the background to light level changes. An active shape tracker is used which takes generated models and attempts to match the contour of the new object to the model. Tracking is performed using a Kalman filter for acceleration and position to predict the future position and then match this with the current frame. The Reading People tracker is built on the Leeds People Tracker. (Siebel, 2000)
6.2.5 The Reading People Tracker

This system consists of a motion detector which feeds a region tracker and head detector. Information from both the region tracker and the head detector are passed to an active shape tracker. Two images follow which broadly describes the operation of the Reading people tracker. (Siebel, 2000)

FIGURE 1 - READING TRACKER MODULE DESCRIPTION – SOURCE: (SIEBEL, 2000, P.32)
FIGURE 2 - READING PEOPLE TRACKER ALGORITHM – SOURCE: (SIEBEL, 2000, P75)
6.3 Motion Detection

The motion detector section of the software is used to determine where movement is occurring in an image. Various filters can be applied and tradeoffs exist between effectiveness of the algorithm and the computational time required for the filter to run. This section will briefly examine the various motion detectors encountered during the literature research with the aim of selecting the most effective combination of filters to provide an adaptive yet real-time and preferably high frame rate system.

6.3.1 Frame Difference Method

This method looks at the difference between this frame and the next in terms of pixel intensity. This method is sensitive to moving background objects such as trees, camera jitter and is sensitive to the threshold chosen.

\[ |(\text{Pixel of Frame})_{\text{now}} - (\text{Pixel of Frame})_{\text{previous}}| > \text{Threshold} \]

6.3.2 Average Filter Method

The background is the average of the last n frames. Differencing and thresholding then follows. Speed and memory consumption are causes for concern with this method.
6.3.3 Median Filter Method

Each pixel is the median of the last n pixel values.

\[ \text{Pixel}_n = \text{Median}(\text{Pixel}_{n-1}, \text{Pixel}_{n-2}, \ldots, \text{Pixel}_{n-l}) \]

\[ l = \text{the length of the median filter}. \]

Absolute differencing then follows between the new background and the new frame and in the event the difference is higher than a threshold, a pixel will be classified as moving. A minor improvement to this method could be the removal of pixels identified as moving from within the median filter buffer. These removed pixels could then be replaced by the last valid background pixel. This method will be sensitive to the threshold value and the length of the buffer. The approximate median filter method obtains a similar quality of result, but is reportedly far more efficient (Velipasalar et al., 2009).

6.3.4 Running Average Method

Background\(_i\) - Background\(_i\) > Threshold

Background\(_i+1\) = \(\alpha\) * Foreground\(_i\) + (1 -\(\alpha\)) * Background\(_i\)

The next background image is equal to a constant (\(\alpha\)) multiplied by the current image plus one – the same system constant multiplied by the current background image.
The older backgrounds have less weight. This method requires low levels of memory as it only stores 2 images for its output. (Velipasalar et al, 2009)

6.3.5 Kalman Filter

A Kalman filter method is used to estimate the background. A Kalman filter predicts the future state of a system and corrects that prediction based on the current measurement. It attempts to identify Gaussian noise with a zero mean and remove it. The optimal state of the process is given by “minimizing the variance of the estimation error and constraining the average of the estimated outputs and the average of the measures to be the same”\(^\text{13}\). The Kalman filter has issues with illumination changes, but low memory requirements and moderate computational complexity (Segata).

6.3.6 Other Filters

Other methods available for background subtraction are Mixtures of Gaussians, Kernel Density Estimators, Mean Shift and Eigenbackgrounds.
6.4 Tracking Methods

6.4.1 Active Shape Tracking

Once a moving region is detected, it’s size and shape are assessed. If it falls within a range of acceptable values, a pedestrian model generated using Principle Component Analysis is scaled and fit to the region. Model fitting is achieved by applying a local edge detector between the difference image of the background and the current image. Estimates are made to find the contour of the person within the region. If the shape matches the model within a given tolerance, the object is said to be a person. A second order motion model is used to predict speed and position in the current frame. Repeated measurements made along the Mahalanobis optimal search direction made at the control points of the B-spline are used to predict future positions.

This method has the advantages of speed and medium robustness. Disadvantages are the inability to detect sitting people, issues with groups of people where individual’s outlines are not clear, edge contrast issues, and tracking initialisation errors. (Siebel, 2000)

6.4.2 Region Tracking

Regions are matched according to their previous size and position to the current size and shape. A first order motion model is used to predict the current position of the
region. A cost function is used to compare the prediction to the current region. (Siebel, 2000)

6.4.3 Mean Shift Tracking

A simplified explanation of mean shift involves determining a histogram for a region of interest. For each frame, around the region of interest, a zone which shows the closest match is then identified as the new position of the tracked object (Wikipedia, 2010).

6.4.4 Feature Tracking

Tracking features of blobs within a motion image and correlating the past and present blobs can provide basic tracking.

Heuristic systems exist where the regions identified after the background subtraction process are classified according to their height and width and the ratio between the two. While this type of system is simple to implement and will track for very basic scenarios, obvious issues will arise during occlusion and times when 2 blobs become joined. Feature tracking however could be used in combination with other methods to improve the abilities of the tracking sections of the program.
The colour components of an identified region can be tracked. It is assumed that the variance between one frame and the next will be relatively low. Once a suitably sized blob is identified, a database entry is made showing a score based on its colour components. A search throughout the entire image is performed to match the last identified object. As each new frame occurs, this score can be updated to account for changes in position.
6.5 Research Summary

Two issues became immediately apparent to the author during the research phase of the project.

The main issue was the high level of prior knowledge assumed with the majority of the systems developed. While most papers were read with interest, much of the theory had not been previously encountered. The most successful trackers were those aimed towards a more educated audience in terms of software engineering and computer vision systems.

The second issue, which also relates to the first was that of time. While many of the more advanced systems would provide better results, the limited time available for the research project means that selecting methods should be done by identifying well documented and simpler methods, although it is acknowledged that the performance of the system may be inferior.

With these considerations in mind the following options were proposed:

1. Develop a mean-shift tracker and attempt to make it track multiple objects.

Limited code is available on the internet, but once again only in C++. The algorithm is reported to be robust and relatively quick however the theory
encountered during the research contained much content not encountered before within the Bachelor of Electrical Engineering.

2. Attempt to compile and modify the Reading People Tracker. This would involve once again acquiring a working knowledge of C++.

3. Use a motion image followed by a feature based tracker which attempts to match the previous regions to the current regions. Use image histograms, size and position as features. Some limited success has been achieved with this approach, but issues such as occlusion will arise (Ali et al, 2010).

The third option will be chosen as it should provide reasonable results for simpler tracking scenarios while being computationally inexpensive, and a working system should be realisable within the allowed project timeframe. This program could act as the foundation for future researchers.
6.6 Initial System Design

An algorithm is proposed and shown below.

![Algorithm Overview](image)

Figure 3 - Algorithm Overview

![Median Filter Diagram](image)

Figure 4 - Approximate Median Filter
USB Camera Pedestrian Counting

It assumed that the quality of the motion image will be acceptable. Some experimentation with post and/or pre-filtering may be required to improve the quality of the motion image. A bounding box will be applied to the blob and a region will be extracted. Some fusing/splitting of adjacent blobs may be performed based on characteristics such as width height ratios and proximity.

It is expected that the feature tracker will be computationally more expensive than the motion image section of the program. For this reason a low resolution grayscale image should be used for feature storage, even though higher resolutions are available. Depending on the processing time, the frame size may be increased. Refer to figure 5 on the following page for the algorithm proposed.

The user interface will have the following features. It will display the run time image with tracking numbers superimposed. It will give the user the ability to modify key variables, select an input source and start/stop the system.
Figure 5 – Initial Feature Tracker Design

MOTION IMAGE INPUT

REGION BOUND BY ZEROS, DETERMINE HEIGHT AND WIDTH OF REGION, ASSUME MINIMUM NUMBER OF PIXELS, ID ALL REGIONS

REGION N - GIVE REGION IDENTIFIER, STORE NO. OF PIXELS, CURRENT X, Y CENTRE, AND REGION ITSELF

COMPARE WITH ALL PREVIOUSLY STORED REGIONS AND MATCH, HISTOGRAM CORRELATION, POSITION CLOSE, SIZE CLOSE

MATCH FOUND? UPDATE REGION PROFILE

CONTINUE TO NEXT REGION

NO MATCH FOUND? MARK AS EXIT, BUT CONTINUE TO STORE FOR N FRAMES

EXIT CONDITIONS, STORED REGION HAS NO MATCH FOUND AFTER N FRAMES
6.7 Programming Language Selection

The authors programming experience was limited to Matlab and programmable logic controllers. Careful selection of a language was needed to ensure that a working product was developed.

A comprehensive review was not performed. Most notably, Java was not trialed. 2 products were primarily investigated to determine the most suitable programming platform. These were Visual Studio 2008 and Matlab Version 7. A free version of Visual Studio 2008 professional was obtained via the Microsoft Dreamspark initiative.

Visual C++ was briefly investigated. There may be a need to call C++ code when speed becomes important. It will be avoided as the learning curve appears steeper. Visual Basic and Matlab provides more managed code, thereby lowering development time.

Image capture using Visual Basic was performed by downloading code snippets. The Webcam was accessed and a bitmap was saved to disk. Some exploration of basic operations such as array manipulation occurred. Visual Studio worked well in all regards but a learning curve of at least 50 hours was expected.
Matlab 7 with the Image Acquisition toolbox was also investigated. Images were acquired, however the frame rate was <10fps. When a frame differencing method was implemented the frame rate dropped to <2fps. It was determined that Matlab would be unsuitable due to its low speed, but could be a good environment for testing Algorithms due to its relative ease of use.

A benchmark performed by OSNews, a programmer orientated website, shows similar performance amongst the more popular languages (2009). It should be noted that details of how the benchmarks conducted were not checked by the author, however, Visual Basic does not drastically lag behind C++ in terms of performance for math operations, although IO operations are significantly lower. It should be noted that C++, C# and Visual Basic .Net framework version compile to a common intermediate language and this may be the reason for the current similarities between execution times of these languages. It is unknown how previous version of Visual Basic prior to the .net framework being used would have fared in terms of speed against C++ and C#.
Some basic code was written in C++, Basic and Matlab to test each platforms time to
completions for a simple for loop which incremented a 32-bit integer. The loop
length is $10^8$. The Windows system clock was used to estimate time to completion.

Matlab Speed Test Code:

```matlab
a = 0
d = 0
length = 100000000
for a = 0:length
    d = d+1;
end
d
```
Visual C++ Speed Test Code

```c
int i;
int b;
for ( i = 0 ; i < 100000000 ; i++ )
    b = b+1;
```

<table>
<thead>
<tr>
<th>Language</th>
<th>Time To Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB</td>
<td>40 seconds</td>
</tr>
<tr>
<td>VBasic.Net</td>
<td>&lt;1 second</td>
</tr>
<tr>
<td>VC++</td>
<td>&lt;1 second</td>
</tr>
</tbody>
</table>

It should be noted that compiler options were set as standard, and that Matlab is significantly faster when it uses intrinsic operations as compared to the extrinsic operations shown in the code snippet, however, intrinsic operations would be rare for the pedestrian tracking application.

Graphedt was also trialled. This software is part of the Microsoft Software Development Kit (SDK). The software can connect to the USB camera using direct X, and then allows the user to write C++ filters and apply them, with the results being placed in a picture box. Unfortunately, this application programming interface (API) did not provide the programmer with the option of creating a user interface, and crashed during initial installations.
Visual Studio 2008, Visual Basic.Net was selected due to its lower learning curve and sufficient speed. Visual Basic is also used within Citect Scada, Allen Bradley PLCs and Excel which are applications currently used by the author.

6.8 Project Resources

Computer Hardware: Quad Core 2.67GHz, i5 750 Processor, 4Gb Ram, Windows 7 64-bit, 9800GT Video Card with 1Gb Ram.

USB Camera: Logitech

Compiler: Microsoft Visual Studio Professional 2008

6.9 Basic Terminology

Pixel – A single square on a screen which is addressable.

Blobs – A contiguous collection of pixels.

Regions – A collection of blobs.

Objects – A collection of regions.
7 DESIGN AND BUILD

7.1 Section Overview

This section will detail the steps taken to arrive at the final version of the pedestrian tracking software. The section headings reflect the path taken during the design and build phase of the project and the issues encountered.

Limited code is included here to show the details of how each stage was accomplished and where that code is not part of the final version of the program. All Visual Basic code is commented to allow those less familiar with the language to grasp the program flow. For each section, program flow diagrams and a written description of the sections purpose is provided.

The author has included this code within the body of this document for 2 primary reasons. Firstly, many tracking systems provide conceptual details, but insufficient detail to implement a working system. By providing this code in a simple language such as Basic, readers will be able to more clearly grasp the detailed steps required. Secondly, the majority of the author’s time has been spent designing and writing the code to provide a working product which clearly demonstrates the results for this type of vision system.
7.2 Design and Build Method

An initial basic design was undertaken by the author during the research phase of the project. During development of the software, the lack of required details shortly became apparent. The approach of the author was to follow the general outline given by the initial design and to grow the missing details. This may be referred to as a top-down approach.

7.3 Image Acquisition

Acquiring an image from the webcam was achieved by using an online tutorial available from [http://www.devx.com/dotnet/Article/30375](http://www.devx.com/dotnet/Article/30375) (Wei Meng, 2010). A windows user form was created, and then the methods outlined in the tutorial were implemented. Some minor modifications occurred as the author did not wish to save video, and only required a single frame captures which are then processed.

Windows media messaging functions and the AviCAP library are used to acquire images. The AviCAP class is a dynamically linked library that provides a message based interface which allows users to access video device drivers. During early phases of the project detailed information regarding the AviCap32.dll was unable to be found. After initial trials with the AviCap32 methods, it was found the resolution and frame rate was sufficient for the needs of the project. Further research has shown that detailed information for the current avicap32.dll can be found within the
Microsoft software developers kit for the .net version 4 framework and from the MSDN website. Other methods would be Direct Show and WIA.

The initial image was captured at 640 by 480 pixels in a 24 bits per pixel RGB format. Frame rate was 30 frames per second. Once the image was acquired the lockbits method (Powell, 2003) was used to place the image data in an array which is included with the drawing.dll. This array had the format of 1 row and 640*480*3 columns and hence 921 600 entries. Indexing for the image throughout the project was difficult due to the 1 dimensional nature of the array. The array represents the pixels values which span from the top left of the screen to the top right, row after row. The table below shows how the data is unwrapped.

**TABLE 1 - PIXEL TO ARRAY MAPPING**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel 1 Red</td>
<td>Pixel 1 Green</td>
<td>Pixel 1 Blue</td>
<td>Pixel 2 Red</td>
<td>Pixel 2 Green</td>
<td>Pixel 2 Blue</td>
<td>Pixel 3 Red</td>
<td>Pixel 3 Green</td>
<td>Pixel 3 Blue</td>
<td>&gt;&gt;&gt;</td>
</tr>
</tbody>
</table>
A formula was developed for finding a particular pixel within the image array in terms of its x,y coordinates. Each pixel has three values.

Pixel(x,y) = x*3-3 + (y-1)*640*3, x*3-2 + (y-1)*640*3, x*3-1 + (y-1)*640*3

If we wish to find pixel (1,1) then the required indices for the image array are...

(0), (1), (2).

If we wish to locate pixel (640, 2) then the required indices for the image array are...

(1917+1920), (1918+1920),(1919+1920).
Some experimentation occurred with the getpixel and setpixel methods to achieve image unwrapping. These are functions from the graphics device interface (GDI) API library. The advantage here is the use of a traditional 2D coordinate system to specify the position of the pixel within the current bitmap. Unfortunately early tests with these functions showed them to be extremely slow and this approach was abandoned.

Once a single image was captured, this process was placed within a while loop and run continuously. This later caused issues with form responsiveness as no time was devoted to checking the windows form for user activity.
Some trials with greyscale conversion were performed to assess the speed improvements and later program motion detection performance.

A typical RGB to grayscale mapping of $0.333^*\text{Red} + 0.59^*\text{Green} + 0.11^*\text{Blue}$ was used. For the interested reader the code is shown below and provides a simple implementation of the required transformation in the VBasic 2010 language.

```vbasic
LockBitmap(newbitmap)

' (0.3333*r+0.59*g+0.11*b)

pix = 0
For Y = 0 To newbitmap.Height - 1
  For X = 0 To newbitmap.Width - 1
    Red = 0.33 * g_PixBytes(pix)
    pix += 1
    Green = 0.59 * g_PixBytes(pix)
    pix += 1
    Blue = 0.11 * g_PixBytes(pix)
    pix += 1
    GrayValue = Math.Floor(Red + Green + Blue)
    If GrayValue > 255 Then
      GrayValue = 255
    End If
    g_PixBytes(pix - 2) = GrayValue
    g_PixBytes(pix - 1) = GrayValue
    g_PixBytes(pix) = GrayValue
  Next X
Next Y

UnlockBitmap(newbitmap)
```

The inner loop ran 307200 times when using a resolution of 640 by 480. Due to the multiplication and floor functions involved there was a significant speed decrease
down to 15 fps. Later trials using greyscale for the motion image also showed that there was no improved performance for the difference image. RGB images were used for the remainder of the project.
7.5 The Background Image

A background image is used by the differencing routine. The idea is that by comparing an empty scene with the current scene, any high level differences are new and moving objects. Some early systems used a static image for the background. This image was acquired while the scene was empty of people. A better approach to creating a background image is by updating the background continuously but omit any moving objects from it. The highly efficient and simple to implement approximate median filter was used.

This approximates the following formula.

\[ \text{Pixel}(i,n) = \text{median} (\text{Pixel}(i,n), \text{Pixel}(i,n-1), \ldots, \text{Pixel}(i,n-l)) \]

\( n = \) the current pixel at TimeNow

\( n-a = \) the current pixel at Time-a

\( l = \) the length of the medium filtered data

The approximate median filter will continue to update the background image over time. The filter performs a pixel by pixel comparison between the background image and the current image which has been captured. If the value of the background pixel is greater than the value of the current image pixel, then the background pixel is
decremented by 1. Similarly, if the background pixel is less than the value of the current image pixel, then the background pixel is incremented by 1. The background image converges towards the values which are the most frequently encountered in the background image. Moving objects cause a temporary disturbance which changes the value of the background image.

The advantage of using this type of filter is that the system can cope with the slow lighting changes which are typical throughout the day. The background image slowly incorporates the new brightness information.

The following table shows a demonstration of the background filter in operation.
USB Camera Pedestrian Counting

TABLE 2 - BACKGROUND IMAGE EXAMPLE

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>The filter is started, the background image is initialised as the current image.</td>
<td>The person moves and the background image remains unchanged due to the high rate of movement.</td>
<td>The person stops moving the hand, and the hand slowly becomes part of the new background.</td>
<td>Once again, the person has stopped moving.</td>
</tr>
</tbody>
</table>

![Original Video](image1)
![Current Background](image2)

![Original Video](image3)
![Current Background](image4)

![Original Video](image5)
![Current Background](image6)

![Original Video](image7)
![Current Background](image8)
7.5.1 The Ghost Filter Variable

The background image update rate affects the entire system. If the frames per second of the system are high and the motion objects moves very slowly through the field of vision, then ghosting occurs. This is a trailing echoed image of the moving object which trails behind the moving object. It is necessary therefore to change the speed at which the background image is updated. To achieve this, the background image is only updated every nth frame. This is referred to as the ghost filter variable within the user application and this variable can be modified while the program is running.

7.6 The Difference Image

The difference image is the difference between the current image and the background image. The current image is subtracted from the background image, the absolute value is found and then a threshold is applied. If pixels in the difference image are above the threshold they are assigned the value of white. If pixels in the difference image are below the threshold value then they are assigned the value of black. White indicates motion, black indicates non-motion.

\[ \text{Pixel}(n) = \text{Pixel}(n_{\text{current}}) - \text{Pixel}(n_{\text{background}}) \]

\[ n = \text{the array index} \]

\[ n = \text{the array index for the current image} \]

\[ n = \text{the array index for the background image} \]
Above in Figure 9 you can see a difference image in the main display window. Shown above the difference image is the current image titled as the original video, and the current background which is empty.
Above in figure 10 you can see the effects of changing the motion threshold variable. In this example the motion variable has been decreased to 60 from the original 120. You can also see in the current background a darkened smudged area which is due to the fast rate at which the median filter is running. This has resulted in the high levels of noise near the primary moving person. Generally however, high levels of noise are apparent throughout the image due to the lower threshold.
FIGURE 11 - DIFFERENCE IMAGE EXAMPLE 3

In the above image, figure 11 you can see in the bottom right hand corner of the difference image the effects of shadows.

Once again, you can also see that the current background has some residual smudging to the left of the moving person caused by the person being in that region for too long.
7.7 The Pixelator

After developing the difference image a concern arose that forming regions from the noisy and sometimes separated regions would be difficult. An example below shows this noise and body part segregation occurring. The hands are clearly distinct from the forearms. The crown of the head has been separated from the face. As it was expected that region growing was to be performed solely by linking those pixels which are white and connected, a method needed to be developed to ensure that pixels of a motion object were joined. A pixelator was written to achieve this.

![FIGURE 12 - SEGREGATION IN THE DIFFERENCE IMAGE](image)

The idea behind the pixelator is to blur and average the image while keeping the computations as low as possible. The method taken was to sweep through all of the horizontal pixels in a blockwise fashion. That is, a row was divided into a number of
blocks. If a given number of pixels in that block were motion pixels, then the entire block was filled with motion pixels. This same method was then applied in the vertical direction. Speed for this method was very high and the results were promising.

In the image below you can see the pixelated image in the main window. The difference image is shown in the top right window. You can see that the hand is separate from the arm and that the arm has two distinct parts, each of them separate from the hand and shoulder. In the pixelated image, the difference image is now one continuous object.

FIGURE 13 - PIXELATOR RESULTS
The pixelator was not used in the final version of the program as the developed region growing method compensated for the image separation which was occurring. The pixelator however showed good speed, but it also changed the boundaries of where the motion was occurring. If later versions of the program were to use contouring this could lead to poor performance due to the boundary shift.
7.7.1 Pixelator Code

'a horizontal pixelate blur is run first

    pix = 0
    For Y = 0 To bmap.Height - 1
        For X = 0 To ((bmap.Width * 3 / BlockSize) - 1)
            For k = 0 To (BlockSize - 1)
                SummedPixels = g_PixBytes(pix) + SummedPixels
                pix += 1
            Next k
            If SummedPixels > BlurThreshold Then
                'set all to motion (255)
                For k = 0 To (BlockSize - 1)
                    g_PixBytes(pix - k - 1) = 255
                Next k
            Else
                'set all to non motion (0)
                For k = 0 To (BlockSize - 1)
                    g_PixBytes(pix - k - 1) = 0
                Next k
            End If
            SummedPixels = 0
        Next X
    Next Y

'a vertical pixelate blur is done next

Dim NumberOfColumns As Integer = bmap.Width * 3
Dim NumberOfRows As Integer = bmap.Height
Dim VertArrayIndex As Integer = 0
Dim NumberOfBlocksPerColumn As Integer = NumberOfRows / BlockSize

Dim VertArray() As Integer
USB Camera Pedestrian Counting

ReDim VertArray(NumberOfRows)

' now setup a column array which lists all the pixel indexes of that column
For X = 0 To (NumberOfColumns - 1)
    VertArray(0) = X
    For k = 1 To NumberOfRows - 1
        VertArray(k) = VertArray(k - 1) + NumberOfColumns
        Next
    pix = 0 'this will count from row number 0 to final row
    ' now process the columns
    For Y = 0 To (NumberOfBlocksPerColumn - 1)
        ' now process the blocks in that column
        For W = 0 To (BlockSize - 1)
            VertArrayIndex = VertArray(pix)
            SummedPixels = g_PixBytes(VertArrayIndex) + SummedPixels
            pix += 1
            Next
        If SummedPixels > BlurThreshold Then
            For Z = 0 To (BlockSize - 1)
                VertArrayIndex = VertArray(pix - 1 - Z)
                g_PixBytes(VertArrayIndex) = 255
                Next
            End If
            SummedPixels = 0
        Next
    Next
7.8 Software Engineering

7.8.1 Multithreading Trials

At this stage the windows form was unresponsive due to the simple for loop which ran the main filter. No time was made for the windows form itself to check if new data was being entered. Multithreading was investigated as this could also lead to significant performance gains.

When checking the windows performance only 27% of the CPU was being used while the program was running. 25% of the CPUs were being used for the running pedestrian tracker application and 2% was being used for the Windows system.

A simple experiment with multi-threading involved placing the form on one thread and the main application on another. This would solve the form’s lack of responsiveness issues and allow the user to click buttons or change variables as required. Multiple issues occurred and these were solved by turning off cross thread call checks and using single thread apartments. Unfortunately a Null Argument Exception continued to occur and this was unable to be debugged. During the period when the program ran successfully the form was immediately responsive.

The final solution to make the form usable was to place a check for `Application.DoEvents()` line of code within the main filter loop. This solution is
not ideal as the form sometimes requires 2 clicks before it will start responding to user input.

7.8.2 Structured Programming

As the project grew it became apparent that the author’s software engineering skills were lacking. The program is essentially several filters running serially. Future versions of the project should ensure that each unique section has been modularized with clearly defined inputs and outputs to allow for easier program development.

7.9 Object Growing

Once a difference image has been generated it is necessary to identify moving objects within the difference image. This is achieved by object growing. The basic concept is to firstly collect all the pixels which are near each other and these are called blobs. Blobs which are in close proximity to one another are then grouped to form regions. Regions which are close to one another are then grouped to form objects. At this stage, no effort is made to detect occlusion. When two people in one scene overlap, this should be later dealt with by region splitting routines or by the use of an omega detector. Object growing simply collects motion pixels which are in close proximity.
In figure 14, how would the pixels be grouped? When a person looks at the image it seems obvious which pixels are a part of the person. How can a program group these pixels? A square has been drawn over the image where the person is. Referring to figure 14 again, one can see that there are blobs which have been separated from the main body in the head and left arm regions.

FIGURE 14 - GROWING OBJECTS FROM A DIFFERENCE IMAGE

The 1st approach investigated was an exhaustive blob growing method which looked for motion pixels in adjacent squares and then grouped them. The 2nd approach was
based on an advanced region growing method as used by a traffic analysis research project.

7.9.1 Region Growing by Seeding

The image could be seeded, and then if the initial seed falls on a motion pixel, then the region search begins.

The above shows a seeded image with a motion region. If the region growing was confined to a square as shown, the resulting image region would be as shown. That is, unless the image was divided using a fine grid, the result would be very blocky. The advantage here is that every pixel does not need to be scanned during the initial sweep. Alternatively, instead of growing the region as a square, once a motion pixel has been identified, then a normal region growing approach could be taken whereby the shape shown above is completely filled.
The above image shows a better method. The image is seeded. If a motion image intersects a seed point as with S2, then a region search begins and any adjacent motion pixels are grouped in that region. S3 to S5 will not initiate any region searches. For S6, a region search is initiated. However, this seed is already part of a region. Before starting the region mapping, S6 is checked to see if it already belongs to a region. If it does, then no region mapping occurs and the program moves onto S7.

The following diagram shows the proposed initial algorithm design.
Another method would be to sweep from left to right through the pixel matrix. When a motion pixel (255) is encountered, start the region map. Once a pixel is added to a region map and the next pixels to be searched are readied, then the pixel is deleted from the original image. The region mapping continues until it can find no more valid motion pixels. Starting at the 1st motion pixel encountered, the sweep continues. This may be faster than the original method and also covers every pixel.

Another option trialed before performing the region growing would be to try and clean the image before processing in the hope of creating larger contiguous regions. Some of these methods are discussed under the pixelator section.
7.9.2 Line by Line Region Growing

Further research lead to a paper entitled “Traffic Image Processing Systems” in which an advanced region growing algorithm was proposed and showed promising speed (Surgailis et al, 2009). The algorithm in this paper is shown below.

This algorithm inspired the approach taken by the author. In essence the following stages occurred in the code developed:

1. Each line was scanned and line blobs were formed.

2. Once the next line had been scanned, both lines were compared.

3. When blobs in Line A overlapped blobs in Line B, then they were merged.
The next line was scanned and the process continued.

![Diagram of line processing stages](image)

**FIGURE 19 - SIMPLIFIED METHOD FOR GROWING REGIONS**

This approach differs from the Advanced Region Growing technique in the fact that a large array is not generated as all lines are not scanned before the blob merging occurs. This method works line by line.

The three stages of line blob formation, line merging and overlapping region growth will now be discussed.

### 7.9.3 Line Blob Formation

The image is currently stored as a 1-dimensional array currently containing 921600 elements. Prior to object growing the 1-dimensional array is reduced by a factor of 3 and now has 307200 elements. This is because this section of the code only needs the X and Y coordinates of each pixel and not all three RGB values. Also, as the difference image effectively flattened the image into a duotone format of 0 for black
and 255 for white, much of the stored information is now redundant. To achieve this, a new array is formed which only uses every third element from the original array.

Scanning from left to right the program groups any blocks of pixels which are in motion (white/255) and adjacent to each other (see figure 19). When a blob is found, its coordinates are saved to a Line Blob array with column headings of Xstart, Xend, Ystart and Yend. Xstart is where the first motion pixel occurs for the line blob. Xend is where the motion pixel of the current blob transitions from motion to non-motion. Ystart and Yend are found by checking which row the program is currently on.

![FIGURE 20 - LINE BLOBS EXAMPLE](image)

The row scanner algorithm is shown in figure 21. The original scanner was modified once further region growing occurred.
7.9.4 Blob Merging

Next blobs of the current line and the previous line are compared and checked if they overlap. A simplified overlap explanation is shown to assist with the conceptual understanding of what operation is being performed in figure 22. If two blobs collide, then they are merged into 1 blob with updated x and y coordinates.

The logic used is shown in figure 23.
Once a line has been scanned, and the merge has occurred, some merged blobs are redundant as they occur within a greater line blob. Hence the current Line blobs are scanned and any redundant blobs are removed from the Line blob array.

### 7.9.5 Region Formation

Next regions must be formed. What happens to blobs on the previous line which have had no matches? When should regions be formed? In order to test these conditions, some basic scenarios were developed and program logic was developed.
USB Camera Pedestrian Counting

The following image shows some of the conditions which would lead to a new region being formed. The regions array contains the coordinates of the region once it has been formed.
FIGURE 24 - REGION GROWTH LOGIC TESTS
Once basic regions had been formed, the regions were then checked to see if they overlapped in a rectangular sense. For example, in the following example, the 2 regions should be combined as they overlap one another. The boxes surrounding each line show the existing coordinates.

![Figure 25 - Region Overlap](image)

A region collision was then performed to combine these two regions. The logic for region collision is similar to the logic used for the line blob overlap, except it also occurs in the y direction. It is also necessary to check region A against region B and region B against region A. Figure 26 shows the 1st case when 2 regions do not overlap, and also the final logic for checking region A to region B. If a collision did occur, new Xstart, Xend, Ystart and Yend boundaries were found for the combined region.
Another piece of logic added to the region growth routine is a check on the size of the region. If the region is too small it is considered to be noise and is deleted from the regions array.

Once region forming had occurred it was found that some regions were very close to each other and it would be sensible to fuse these regions. An option was added to the program called object minimum distance. If 2 regions were close to each other and within the minimum distance in any x or y direction, then they should be fused into a greater region. This was achieved by growing a region in all directions by the minimum distance value. Once this has been done for all regions, a collision detect and merge was once again performed.
During this development process a self contained program was written to assist with debugging. Figure 27 shows the program and the generated arrays when object growth is set to zero, and hence no object expansion and collision detection occurs. Figure 28 shows the results when the objects are expanded and then the collision detection occurs. It is obvious there are some deficiencies with this method as in figure 28 some objects are combined which it would be preferable not to combine. The selection of an appropriate region minimum distance value is needed.

FIGURE 27 - REGION GROWTH WITH NO OBJECT OVERLAP
7.9.6 Region Growing Results

Once the object growing had been incorporated into the main program the following results were obtained.
USB Camera Pedestrian Counting

FIGURE 29 - OBJECTS EXAMPLE 1

FIGURE 30 - OBJECTS EXAMPLE 2
Figure 31 shows that issues were being caused by shadows. A shadow occurs on the left of the subject on the near wall. These shadows lead to a significantly larger object being drawn than was actually occurring. This problem became more apparent when outdoor tests were performed as demonstrated in the section on tracking. In order to remove these shadows, a suitable environment would need to be chosen, or shadow removal techniques would have to be developed.

The user interface developed gives the option of running the program using a lower resolution. Figure 32 shows that the results are similar. The processing time when moving to a 160 by 120 image are significantly faster with frame rates approaching 31fps. However, there is no significant gain when working at 320 by 240 resolution.
with frame rates of 29 fps. Compare this to the normal program speed of approximately 28 fps. This is due to the time required for the resize calculation itself. This could be improved however if the software had the ability to control the camera driver directly and set the camera image format at the required resolution.

FIGURE 32 - OBJECTS EXAMPLE LOWER RESOLUTION
7.10 Basic Object Tracking

Unfortunately due to time constraints this section needed to be simplified in order to provide some results which could be tested. As such, the tracking techniques were insufficient for any complex situation where multiples objects appear.

Object tracking used 3 basic premises.

1. Objects which have a similar size could be related.

2. Objects which have a similar position could be related.

3. Use the age of the matched objects to gain or lose a track.

The code considered all of the current objects in terms of size and position, and compared this to all of the objects from the previous frame. If a match was found, then the object was placed in a possible objects array and the match found counter was incremented for this possible object. If this possible object had a high match, then it was tracked. As new frames arrived and new objects occurred, if these new objects did not match the possible objects, the match found counter was decremented.
USB Camera Pedestrian Counting

The result was promising in a simple environment where there was only one primary object. It was expected that the outdoor results would not be as successful.

FIGURE 33 – INDOOR TRACKING EXAMPLE 1
Outdoor results demonstrated many of the shortcomings of the approach taken by the author. These included tracker confusion when two objects overlapped and object distortion when shadows were present. Some of these results are given in the following figures.
Figure 35 shows how moving trees and shadows cause issues with the program. The person in the top centre of the picture has not been tracked.
In figure 36, the tracker now thinks there is 2 objects in the 1 region. In figure 37 it can be seen that the tracker was started while a person was in the frame. As such, there exists an impression of them within the median filtered image. Note better results were gained depending on the time of day.
USB Camera Pedestrian Counting

FIGURE 37 - TRACKING OUTDOORS GOOD RESULTS

FIGURE 38 - TRACKING OUTDOORS OCCLUSION
7.11 Final Code

While the code for a research project is typically included as an appendix, the author’s efforts have been primarily directed towards producing a working software application. This project has been primarily a work in software engineering and vision systems. As such, the final version of the software is given here in its entirety and it is hoped that this may be used by future students or researchers. This code provides a practical realisation of the vision systems theory. Visual Basic project files, of which there are multiple versions, are also available from jezzalanka@hotmail.com upon request.

It should be noted that some code has been used which is freely available from the internet. In particular, the lockbits method (Powell, 2003) and the main image capture routine (Wei Meng, 2009) has been taken from online programming tutorials. This code amounts to less than 5% of the total code compiled by the author.
FIGURE 39 - WINDOWS FORM DESIGN
 Imports System.Runtime.InteropServices
 Imports System.Drawing
 Imports System.Drawing.Graphics
 Imports System.Threading

 Public Class Form1

 'these are constants used for image capture

 Const WM_CAP_START = &H400S
 Const WS_CHILD = &H40000000
 Const WS_VISIBLE = &H10000000

 Const WM_CAP_DRIVER_CONNECT = WM_CAP_START + 10 'creates a child window
 Const WM_CAP_DRIVER_DISCONNECT = WM_CAP_START + 11 'creates a window that is initially visible

 Const WM_CAP_EDIT_COPY = WM_CAP_START + 30 'connects a capture window to a capture driver.
 Const WM_CAP_SEQUENCE = WM_CAP_START + 62 'disconnects a capture driver from a capture window
 Const WM_CAP_FILE_SAVEAS = WM_CAP_START + 23 'copies video frame buffer to the clipboard
 Const WM_CAP_SET_SCALE = WM_CAP_START + 53 'initiates streaming capture to a file
 Const WM_CAP_SET_PREVIEWRATE = WM_CAP_START + 52 'copies the contents of the capture file to another file
 Const WM_CAP_SET_PREVIEW = WM_CAP_START + 50 'enables or disables scaling of the preview video images

 Const SWP_NOMOVE = &H2S 'sets the frame display rate in preview mode
 Const SWP_NOSIZE = 1 'enables or disables preview mode.
 Const SWP_NOZORDER = &H4S 'changes the size, position, and Z order of a child, pop-up, or top-level window
 Const HWND_BOTTOM = 1 'retains the current size (ignores the cx and cy parameters).
 Const HWND_TOPMEN = &H4S 'retains the current Z order (ignores the hWndInsertAfter parameter).
 Const HWND_TOP = 1 'places the window at the bottom of the Z order. If the hWnd parameter identifies a topmost window, the window loses its topmost status and is placed at the bottom of all other windows.

 Dim RUN_SYSTEM As Integer
USB Camera Pedestrian Counting

```
Dim FRAME_RATE_COUNTER As Long
Dim Red, Green, Blue As Integer
Dim ChangeThreshold As Integer
Dim Image_Size As Integer = 1

'state constants used by the object building section of code

Dim Length As Integer = 300
  'the length of the following arrays - too small and the program will crash
Dim LineA(4, Length) As Integer
  'the previous line blob array (Xstart, Xend, Ystart, Yend)
Dim LineB(4, Length) As Integer
  'the current line blob array (Xstart, Xend, Ystart, Yend)
Dim LineTemp(4, Length) As Integer
  'a temp storage line blob array (Xstart, Xend, Ystart, Yend) - these are the new LineA values
Dim Regions(4, Length) As Integer
  'the stored regions array (Xstart, Xend, Ystart, Yend)
Dim Xstart As Integer = 0
  'the start pixel of the current blob/region
Dim Xend As Integer = 0
  'the last pixel of the current blob/region
Dim Ystart As Integer = 0
  'the start row of the current region/region
Dim Yend As Integer = 0
  'the end row of the current region/region
Dim CurrentRow As Integer = 0
  'the current row number
Dim CurrentColumn As Integer = 0
  'the current column number
Dim RowStartPixel As Integer = 0
  'the start pixel number of the current row
Dim RowEndPixel As Integer = 0
  'the end pixel number of the current row
Dim Pixel As Integer = 0
  'the current pixel number
Dim PixelsPerRow As Integer = 0
  'number of pixels per row
Dim NumberOfPixels As Integer = 0
  'the accumulated number of pixels for the current line blob
Dim NumberOfRows As Integer = 0
  'the number of rows in the current image
Dim NumberOfColumns As Integer = 0
  'the number of columns in the current image
Dim NewRow As Integer = 0
  'do we need to start a new row?
Dim LastPixel As Integer = 0
  'was the last pixel checked a motion pixel
Dim Objects(4, Length) As Integer
  'the stored objects array (Xstart, Xend, Ystart, Yend)
Dim NewRegions(4, Length) As Integer
  'the new stored regions array (Xstart, Xend, Ystart, Yend)
Dim BlobNumber As Integer = 0
  'the current blob Number - used as a pointer the LineB array
Dim RegionMinSize As Integer = 10
  'the minimum size of a region - note total p1xles not calculated, only total height + length
Dim RegionMinimumDistance As Integer = 5
  'this is used to determine if 2 regions overlap
Dim BlobNumberB As Integer = 0
  'the total number of blobs in LineB, starts at zero
Dim BlobNumberA As Integer = 0
  'the total number of blobs in LineA, starts at zero
Dim BlobNumberTemp As Integer = 0
  'the total number of blobs in LineTemp, starts at zero
```
USB Camera Pedestrian Counting

Dim RegionsNumber As Integer 'the region array pointer
Dim BlobB As Integer = 0 'a blob B pointer
Dim BlobA As Integer = 0 'a blob A pointer
Dim BlobTemp As Integer = 0 'a blob temp pointer
Dim BlobMerge As Integer = 0 'continue merging for this current blob?
Dim MatchFound As Integer = 0 'this creates a region with the current LineA Blob
Dim BlobPointer As Integer = 0 'add the BlobB pointer to the BlobMerge pointer
Dim AllRegionsChecked As Integer = 0 'used to merge the regions array
Dim BorderPixelsThree As Integer = 0 '3 time multiply
Dim RowNumber As Integer = 0 'the current row number
Dim RowNumberStart As Integer = 0 'the start row number
Dim RowNumberEnd As Integer = 0 'the end row number
Dim RegionLength As Integer = 0 'the length of the current region
Dim RegionHeight As Integer = 0 'the height of the current region
Dim RegionSize As Integer = 0 'the total of the height and length of a region
Dim NewRegionsPointer As Integer = 0 'a pointer for the new regions array
Dim NewRegionsNumber As Integer = 0 'the number of new entries in the newregions array
Dim RegionMatchFound As Integer = 0 'indicates a match has been found between 2 regions
Dim RegionsMatchPointer As Integer = 0 'a region match pointer
Dim ObjectsNumber As Integer = 0 'The total number of objects in the objects array
Dim PossibleObjects (?, Length) As Integer 'size, xcentre, ycentre, number of matches, match found this iteration, delete this entry, ID
Dim PossibleObjectsTemp (?, Length) As Integer 'size, xcentre, ycentre, number of matches, match found this iteration, delete this entry, ID
Dim PossibleObjectsNumber As Integer 'the number of current possible objects
Dim ObjectID As Integer = 0 'the Object ID
Dim ObjectsMatchStatus (?, Length) As Integer 'the number of pedestrians which have crossed since program start

'these constants are using for the drawing
Dim Pen As New Pen(Color.FromArgb(255, 0, 255, 0), 3)
Dim drawFont As New Font("Arial", 40)

'ghost filter constant
Dim GhostFilter As Integer = 1
'try setting label near start to ensure it displays on form load

'--The capGetDriverDescription function retrieves the version
' description of the capture driver--
Declare Function capGetDriverDescriptionA Lib "avicap32.dll" _
    (ByVal wDriverIndex As Short, _
    ByVal lpszName As String, ByVal cbName As Integer, _
    ByVal lpszVer As String, _
    ByVal cbVer As Integer) As Boolean

'--The capCreateCaptureWindow function creates a capture window--
Declare Function capCreateCaptureWindowA Lib "avicap32.dll" _
    (ByVal lpszWindowName As String, ByVal dwStyle As Integer, _
    ByVal x As Integer, ByVal y As Integer, ByVal nWidth As Integer, _
    ByVal nHeight As Short, ByVal hWnd As Integer, _
    ByVal nID As Integer) As Integer

'--This function sends the specified message to a window or windows--
Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
    (ByVal hwnd As Integer, ByVal Msg As Integer, _
    ByVal wParam As Integer, _
    <MarshalAs(UnmanagedType.AsAny)> ByVal lParam As Object) As Integer

'--Sets the position of the window relative to the screen buffer--
Declare Function SetWindowPos Lib "user32" Alias "SetWindowPos" _
    (ByVal hwnd As Integer, _
    ByVal hWndInsertAfter As Integer, ByVal x As Integer, _
    ByVal y As Integer, _
    ByVal cx As Integer, ByVal cy As Integer, _
    ByVal wFlags As Integer) As Integer

'--This function destroys the specified window--
Declare Function DestroyWindow Lib "user32" _
    (ByVal hndw As Integer) As Boolean

'---used to identify the video source---
Dim VideoSource As Integer
'---used as a window handle---
Dim hWnd As Integer

'---preview the selected video source---
Private Sub PreviewVideo(ByVal pbCtrl As PictureBox)
    hWnd = capCreateCaptureWindowA(VideoSource, _
        WS_VISIBLE Or WS_CHILD, 0, 0, 0, _
        0, pbCtrl.Handle.ToInt32, 0)
    If SendMessage( _
        hWnd, WM_CAP_DRIVER_CONNECT, _
        VideoSource, 0) Then
        '---set the preview scale---
        SendMessage(hWnd, WM_CAP_SET_SCALE, True, 0)
        '---set the preview rate (ms)---
        SendMessage(hWnd, WM_CAP_SET_PREVIEWRATE, 10, 0)
        '---start previewing the image---
        SendMessage(hWnd, WM_CAP_SET_PREVIEW, True, 0)
        '---resize window to fit in PictureBox control---
        SetWindowPos(hWnd, HWND_BOTTOM, 0, 0, _
            pbCtrl.Width, pbCtrl.Height, _
            SWP_NOMOVE Or SWP_NOZORDER)
    Else
        '---error connecting to video source---
        DestroyWindow(hWnd)
    End If
End Sub

'---stop the preview window---
Private Sub btnStopCamera_Click(_
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
Handles btnStop.Click
    StopPreviewWindow()
End Sub

'--disconnect from video source---
Private Sub StopPreviewWindow()
sendMessage(hWnd, WM_CAP_DRIVER_DISCONNECT, VideoSource, 0)
DestroyWindow(hWnd)
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    '---list all the video sources---
    ListVideoSources()
End Sub

'---list all the various video sources---
Private Sub ListVideoSources()
    Dim DriverName As String = Space(80)
    Dim DriverVersion As String = Space(80)
    For i As Integer = 0 To 9
        If capGetDriverDescriptionA(i, DriverName, 80, _
            DriverVersion, 80) Then
            lstVideoSources.Items.Add(DriverName.Trim)
        End If
    Next
End Sub

Private Sub PictureBox1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PictureBox1.Click
End Sub

'---list all the video sources---
Private Sub lstVideoSources_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles lstVideoSources.SelectedIndexChanged
    'stop all existing previews and filters to prevent program crash
    RUN_SYSTEM = 0
    StopPreviewWindow()
---check which video source is selected---
VideoSource = lstVideoSources.SelectedIndex
---preview the selected video source
PreviewVideo(PictureBox1)
End Sub

Private Sub btnPreviewWindow_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnStop.Click
    ---stop the preview window---
    StopPreviewWindow()
End Sub

'-------------------------------------------
'Process Image
'-------------------------------------------

'---save the image---
Private Sub ProcessImage()

    Dim data As IDataObject
    Dim bmap As Image 'original captured full colour image
    Dim diffbmap As Image 'the motion image
    Dim medbmap As Image 'the medium filtered image
    Dim X, Y, k, pix As Integer
    Dim ArrayLength As Integer
    Dim GhostCounter As Integer
    GhostCounter = 0

    '---copy the current preview image to the clipboard---
    SendMessage(hWnd, WM_CAP_EDIT_COPY, 0, 0)

    '---retrieve the image from clipboard and convert it ' to the bitmap format
    data = Clipboard.GetDataObject()
    If data.GetDataPresent(GetType(System.Drawing.Bitmap)) Then
        bmap = _
USB Camera Pedestrian Counting

```
    CType(data.GetData(GetType(System.Drawing.Bitmap)), _ Image)
End If

If Image_Size = 2 Then
    'now lower the size to speed processing up, Avicap appears limited in size options
    'after including this, speed increase was not significant, probably due to added rescale time
    bmap = ResizeImage(bmap, 0.5, 0.5)
End If

If Image_Size = 4 Then
    'now lower the size to speed processing up, Avicap appears limited in size options
    'after including this, speed increase was not significant, probably due to added rescale time
    bmap = ResizeImage(bmap, 0.25, 0.25)
End If

'the following initialises the bitmap arrays used in later modules
LockBitmap(bmap)

Dim background() As Byte
ArrayLength = g_PixBytes.GetLength(0)
ReDim background(ArrayLength)
background = g_PixBytes

Dim DiffArray() As Byte
ReDim DiffArray(ArrayLength)

UnlockBitmap(bmap)

'these are used for the conversion to grayscale process
Dim AbsoluteValue, AbsoluteValueRed, AbsoluteValueBlue, AbsoluteValueGreen As Integer
```
Dim CurrentBackgroundPixel As Integer
Dim CurrentPixel As Integer

'set the default motion threshold
ChangeThreshold = 60

'establish some constants
PixelsPerRow = bmap.Width * 3
NumberOfRows = bmap.Height
RowEndPixel = RowStartPixel + PixelsPerRow

'display some info about the system while building system
Label2.Text = "Image Height = " + CStr(bmap.Height)
Label3.Text = "Image Width = " + CStr(bmap.Width)
Label4.Text = "Array Length = " + CStr(ArrayLength)
Label5.Text = CStr(ChangeThreshold)
Label18.Text = CStr(GhostFilter)
Label16.Text = CStr(RegionMinSize)
Label17.Text = CStr(RegionMinimumDistance)

While RUN_SYSTEM = 1

'---copy the current preview image to the clipboard---
SendMessage(hWnd, WM_CAP_EDIT_COPY, 0, 0)

'---retrieve the image from clipboard and convert it 'to the bitmap format
data = Clipboard.GetDataObject()
If data.GetDataPresent(GetType(System.Drawing.Bitmap)) Then

  bmap = _
    CType(data.GetData(GetType(System.Drawing.Bitmap)), _
      Image)
  medbmap = _
    CType(data.GetData(GetType(System.Drawing.Bitmap)), _
      Image)
diffbmap = _
CType(data.GetData(GetType(System.Drawing.Bitmap)), _
Image)

'reduce image size if 320*240 mode selected
If Image_Size = 2 Then
    bmap = ResizeImage(bmap, 0.5, 0.5)
    medbmap = ResizeImage(medbmap, 0.5, 0.5)
    diffbmap = ResizeImage(diffbmap, 0.5, 0.5)
End If

'reduce image size if 160*120 mode selected
If Image_Size = 4 Then
    bmap = ResizeImage(bmap, 0.25, 0.25)
    medbmap = ResizeImage(medbmap, 0.25, 0.25)
    diffbmap = ResizeImage(diffbmap, 0.25, 0.25)
End If

'------------------------------------------------
'Background Image Creation
'apply the median filter and then display the image in picture box 3

' Lock the bitmap data.
LockBitmap(medbmap)

'lockbit returns an array g_PixBytes with a format of r1,g1,b1,r2,g2,b2...
'bitmap is drawn from top left to right, line by line

to find a particular element in the array use the following formula
'Given X and Y coordinates, the address of the first element in the
'pixel is (y*Stride)+(x*3).
'This points to the blue byte which is followed by the green and the red.

'generate the background image using an approximate median filter
'only do every nth iteration otherwise ghosting occurs
USB Camera Pedestrian Counting

If GhostCounter > 100 Then
  GhostCounter = 0
End If

If GhostCounter = GhostFilter Then
  pix = 0
  For Y = 0 To bmap.Height - 1
    For X = 0 To bmap.Width - 1
      For k = 0 To 2
        If background(pix) < g_PixBytes(pix) Then
          background(pix) = background(pix) + 1
        ElseIf background(pix) > g_PixBytes(pix) Then
          background(pix) = background(pix) - 1
        End If
        pix += 1
      Next k
    Next X
  Next Y
  g_PixBytes = background
  UnlockBitmap(medBmap)
  'display the background image
  PictureBox3.Image = medBmap

  'this is used to slow down the median filter and only affects the background image
  GhostCounter = 0
Else
  g_PixBytes = background
  UnlockBitmap(medBmap)
  PictureBox3.Image = medBmap
  GhostCounter += 1
End If

'-------------------------------------------
'Difference Image
LockBitmap(diffbmap)
'next the differencing will occur which will result in the final motion image

pix = 0
For Y = 0 To bmap.Height - 1
    For X = 0 To bmap.Width - 1
        CurrentBackgroundPixel = background(pix)
        CurrentPixel = g_PixBytes(pix)
        AbsoluteValueRed = Math.Abs(CurrentPixel - CurrentBackgroundPixel)
        pix += 1
        CurrentBackgroundPixel = background(pix)
        CurrentPixel = g_PixBytes(pix)
        AbsoluteValueBlue = Math.Abs(CurrentPixel - CurrentBackgroundPixel)
        pix += 1
        CurrentBackgroundPixel = background(pix)
        CurrentPixel = g_PixBytes(pix)
        AbsoluteValueGreen = Math.Abs(CurrentPixel - CurrentBackgroundPixel)
        pix += 1
        AbsoluteValue = AbsoluteValueRed + AbsoluteValueBlue + AbsoluteValueGreen
        If AbsoluteValue < ChangeThreshold Then
            g_PixBytes(pix - 2) = 0
            g_PixBytes(pix - 1) = 0
            g_PixBytes(pix) = 0
        Else
            g_PixBytes(pix - 2) = 255
            g_PixBytes(pix - 1) = 255
            g_PixBytes(pix) = 255
        End If
        pix += 1
    Next X
Next Y

DiffArray = g_PixBytes

' Unlock the bitmap data.
UnlockBitmap(diffbmap)

'display the motion image
PictureBox4.Image = diffbmap

'------------------------------------------------
'Object Building
'------------------------------------------------

g_PixBytes = DiffArray

'Reduce the array size to speed up processing.

Dim Reduced_g_PixBytes() As Byte
ReDim Reduced_g_PixBytes(bmap.Height * bmap.Width)

Dim Pix1 As Integer = 0
Dim Pix3 As Integer = 0

For k = 0 To bmap.Height * bmap.Width - 1
    Reduced_g_PixBytes(Pix1) = g_PixBytes(Pix3)
    Pix1 += 1
    Pix3 += 3
Next k

'section constants
ReDim LineA(4, Length) 'the previous line blob array
(0, Xstart, Xend, Ystart, Yend, NumberOfPixels)
ReDim LineB(4, Length) 'the current line blob array
(0, Xstart, Xend, Ystart, Yend, NumberOfPixels)
ReDim LineTemp(4, Length) 'a temp storage line blob array
(Xstart, Xend, Ystart, Yend, NumberOfPixels)
USB Camera Pedestrian Counting

ReDim Regions(4, Length) 'the stored regions array
(Regions, Xstart, Xend, Ystart, Yend, NumberOfPixels)
ReDim Objects(4, Length) 'the stored objects array (Xstart, Xend, Ystart, Yend)
ReDim NewRegions(4, Length) 'the new stored regions array (Xstart, Xend, Ystart, Yend)
Xstart = 0 'the start pixel of the current blob/region
Xend = 0 'the last pixel of the current blob/region
Ystart = 0 'the start row of the current region/region
Yend = 0 'the end row of the current region/region
CurrentRow = 0 'the current row number
CurrentColumn = 0 'the current column number
RowStartPixel = 0 'the start pixel number of the current row
RowEndPixel = 0 'the end pixel number of the current row
BlobNumber = 0 'the current blob Number - used as a pointer the LineB array
Pixel = 0 'the current pixel number
PixelsPerRow = 0 'the accumulated number of pixels for the current line blob
NumberOfPixels = 0 'the number of rows in the current image
NewRow = 0 'do we need to start a new row?
LastPixel = 0 'was the last pixel checked a motion pixel
BlobNumberB = 0 'the total number of blobs in LineB, starts at zero
BlobNumberA = 0 'the total number of blobs in LineA, starts at zero
BlobNumberTemp = 0 'the total number of blobs in LineTemp, starts at zero
RegionsNumber = 0 'the region array pointer
BlobB = 0 'a blob B pointer
BlobA = 0 'a blob A pointer
BlobTemp = 0 'a blob temp pointer
BlobMerge = 0 'continue merging for this current blob?
MatchFound = 0 'this creates a region with the current LineA Blob
BlobPointer = 0 'add the BlobB pointer to the BlobMerge pointer
AllRegionsChecked = 0 'used to merge the regions array
BorderPixelsThree = 0 '3 time multiply
RowNumber = 0 'the current row number
RowNumberStart = 0 'the start row number
RowNumberEnd = 0 'the end row number
RegionLength = 0 'the length of the current region
RegionHeight = 0 'the height of the current region
RegionSize = 0 'the total of the height and length of a region
NewRegionsPointer = 0 'a pointer for the new regions array
USB Camera Pedestrian Counting

NewRegionsNumber = 0 'the number of new entries in the newregions array
RegionMatchFound = 0 'indicates a match has been found between 2 regions
RegionsMatchPointer = 0 'a region match pointer
ObjectsNumber = 0 'The total number of objects in the objects array

'scan a row

BlobNumber = 0
LastPixel = 0
RowStartPixel = 0

'LockBitmap(regionbmap)

NumberOfRows = bmap.Height
NumberOfColumns = bmap.Width
Pixel = 0

g_PixBytes = DiffArray

'delete the last line of the reduced array to prevent regions froms being missed

RowStartPixel = (NumberOfRows - 1) * NumberOfColumns
RowEndPixel = NumberOfRows * NumberOfColumns - 1

For Pixel = RowStartPixel To RowEndPixel - 1
    Reduced_g_PixBytes(Pixel) = 0
Next

PixelsPerRow = NumberOfColumns

'end reduce
RowStartPixel = 0

For CurrentRow = 0 To NumberOfRows - 1
    'the following find blobs in a line and groups them.
    'the array lineB then contains there start and stop positions.
RowEndPixel = RowStartPixel + PixelsPerRow
BlobNumberB = 0
LastPixel = 0
CurrentColumn = 0

'clear the LineB array before starting the pixel search

For i = 0 To 3
    For j = 0 To Length - 1
        LineB(i, j) = 0
    Next
Next

For Pixel = RowStartPixel To RowEndPixel - 1
    If Reduced_g_PixBytes(Pixel) = 0 And LastPixel = 0 Then
        LastPixel = 0
    ElseIf Reduced_g_PixBytes(Pixel) = 255 And LastPixel = 0 Then
        BlobNumberB += 1
        'blob number increments on positive edge, blob 0 always zero.
        LineB(0, BlobNumberB) = CurrentColumn  '(Xstart,Xend,Ystart,Yend)
        LineB(2, BlobNumberB) = CurrentRow
        LineB(3, BlobNumberB) = CurrentRow
        LastPixel = 1
    ElseIf Reduced_g_PixBytes(Pixel) = 255 And LastPixel = 1 Then
        LineB(1, BlobNumberB) = CurrentColumn  '(Xstart,Xend,Ystart,Yend)
    ElseIf Reduced_g_PixBytes(Pixel) = 0 And LastPixel = 1 Then
        LastPixel = 0
    End If

    CurrentColumn += 1
Next

'if LineA, BlobA has no matches with any of LineB blobs then it must be a region
'it is then copied to the regions array.

If BlobNumberB = 0 Then
    If BlobNumberA > 0 Then
        For BlobA = 1 To BlobNumberA
            'update the regions array
            If LineA(1, BlobA) - LineA(0, BlobA) > RegionMinSize Then
                For i = 0 To 3
                    Regions(i, RegionsNumber) = LineA(i, BlobA)
                Next i
                RegionsNumber += 1        'the Regions array pointer
            End If
        Next
    End If
End If

BlobNumberA = 0

End If

'does a blob exist in lineB yet? No, then skip all of this processing!

If BlobNumberB > 0 Then

    'this section will compare LineA(previous line) to LineB(current line) and update LineB.
    'result will be an updated LineB array.

    For BlobA = 1 To BlobNumberA
        For BlobB = 1 To BlobNumberB


If (LineB(1, BlobB) >= LineA(0, BlobA) And LineB(0, BlobB) <= LineA(1, BlobA)) = 0 Then

'scan the next Blob of LineB instead as there is no match
'this will be the most common case
MatchFound = 0

Else

'a match is found. update lineB with LineA blobcurrent info
MatchFound = 1

'what is the new Xend value for LineB, BlobB?
If LineA(1, BlobA) >= LineB(1, BlobB) Then
  LineB(1, BlobB) = LineA(1, BlobA)
End If

'what is the new Xstart value for LineB, BlobB?
If LineB(0, BlobB) >= LineA(0, BlobA) Then
  LineB(0, BlobB) = LineA(0, BlobA)
End If

'what is the new YStart value for LineB, BlobB?
If LineB(2, BlobB) >= LineA(2, BlobA) Then
  LineB(2, BlobB) = LineA(2, BlobA)
End If

Exit For

End If

Next BlobB

If MatchFound = 0 Then

'this blob must be a new region. Update the region array.
'a filter needs to be added here to remove noise

If LineA(1, BlobA) - LineA(0, BlobA) > RegionMinSize Then
USB Camera Pedestrian Counting

For i = 0 To 3
    Regions(i, RegionsNumber) = LineA(i, BlobA)
Next i

RegionsNumber += 1 'the Regions array pointer

End If

End If

Next BlobA

'Now Update LineA with a LineB which has had all overlapping blobs removed.

For i = 0 To 3
    For j = 0 To Length - 1
        LineA(i, j) = 0
    Next
Next

BlobNumberA = 0 BlobMerge = 1 'this is set equal to the start entry of LineB

'set BlobB + 1 entry out of bounds so that when last blob is scanned, 'match won't be found if the 1st blob starts at 0,0

For i = 0 To 1
    LineB(i, BlobNumberB + 1) = 10000
Next i

For BlobB = 1 To BlobNumberB

    BlobMerge = 1
    While BlobMerge > 0

        BlobPointer = BlobB + BlobMerge

        If (LineB(1, BlobB) >= LineB(0, BlobPointer) And LineB(0, BlobB) <= LineB(1, BlobPointer)) Then
'now merge next blob with current blob
'what is the new Xend value for LineB, BlobB?
If LineB(1, BlobB + BlobMerge) >= LineB(1, BlobB) Then
  LineB(1, BlobB) = LineB(1, BlobB + BlobMerge)
End If

'what is the new Xstart value for LineB, BlobB?
If LineB(0, BlobB + BlobMerge) <= LineB(0, BlobB) Then
  LineB(0, BlobB) = LineB(0, BlobB + BlobMerge)
End If

'what is the new Ystart value for LineB, BlobB?
If LineB(2, BlobB + BlobMerge) <= LineB(2, BlobB) Then
  LineB(2, BlobB) = LineB(2, BlobB + BlobMerge)
End If

BlobMerge += 1

Else
'no matches have been found for the current blob within the next blob.
'this then is the newest entry for LineA

BlobNumberA += 1

For i = 0 To 3
  LineA(i, BlobNumberA) = LineB(i, BlobB)
Next i

BlobB = BlobB + BlobMerge - 1
BlobMerge = 0

End If

End While

Next BlobB
End If

' prep for the next loop
RowStartPixel = RowStartPixel + PixelsPerRow

Next CurrentRow

' now group the regions if they overlap
' 1st, delete any regions which are not the minimum size. This should remove regions created
due to noise

For RegionsPointer = 0 To RegionsNumber - 1

' find how many pixels in the current region
RegionLength = Regions(1, RegionsPointer) - Regions(0, RegionsPointer)  ' xend - xstart
RegionHeight = Regions(3, RegionsPointer) - Regions(2, RegionsPointer)  ' yend - ystart
RegionSize = RegionLength + RegionHeight

If RegionSize >= RegionMinSize Then

' copy this region to the RegionTemp array and increment the regiontempnumber counter
For i = 0 To 3
    NewRegions(i, NewRegionsNumber) = Regions(i, RegionsPointer)
Next i

NewRegionsNumber += 1

End If

Next

' now clear the old regions array and replace with the newregions

For RegionsPointer = 0 To RegionsNumber - 1
    For i = 0 To 3
        Regions(i, RegionsPointer) = 0
    Next i
Next
For NewRegionsPointer = 0 To NewRegionsNumber
    For i = 0 To 3
        Regions(i, NewRegionsPointer) = NewRegions(i, NewRegionsPointer)
    Next i
Next

RegionsNumber = NewRegionsNumber
NewRegionsPointer = 0

'now, we want to grow the region by a number of pixels in all directions and then do a collision detect.
'if they collide after the growth, they are in close proximity
'we will add RegionsMinimumDistance, or subtract as necessary to each of our coordinates.
'care must be taken not to exceed the boundaries of the current image

For RegionsPointer = 0 To RegionsNumber - 1
    Regions(0, RegionsPointer) = Regions(0, RegionsPointer) - RegionMinimumDistance
    Regions(1, RegionsPointer) = Regions(1, RegionsPointer) + RegionMinimumDistance
    Regions(2, RegionsPointer) = Regions(2, RegionsPointer) - RegionMinimumDistance
    Regions(3, RegionsPointer) = Regions(3, RegionsPointer) + RegionMinimumDistance
    If Regions(0, RegionsPointer) < 0 Then
        Regions(0, RegionsPointer) = 0
    End If
    If Regions(1, RegionsPointer) > PixelsPerRow - 1 Then
        Regions(1, RegionsPointer) = PixelsPerRow - 1
    End If
    If Regions(2, RegionsPointer) < 0 Then
        Regions(2, RegionsPointer) = 0
    End If
    If Regions(3, RegionsPointer) > NumberOfRows - 1 Then
        Regions(3, RegionsPointer) = NumberOfRows - 1
    End If
Next

' now we will compare all regions to all regions.
' merging will occur

' 1st add 2 final entries which will mean all compares are out bounds to end program properly

For k = 0 To 1
    For i = 0 To 3
        Regions(i, RegionsNumber + k) = 100000
    Next i
Next k

Dim RegionEnd As Integer = 0

For RegionsPointer = 0 To RegionsNumber - 1

    ' xoverlap (current to next)
    ' Regions(1, RegionsPointer + 1) >= Regions(0, RegionsPointer) And Regions(0, RegionsPointer + 1) <= Regions(1, RegionsPointer)
    ' yoverlap (current to next)
    ' Regions(3, RegionsPointer + 1) >= Regions(2, RegionsPointer) And Regions(2, RegionsPointer + 1) <= Regions(3, RegionsPointer)

    ' xoverlap (next to current)
    ' Regions(1, RegionsPointer) >= Regions(0, RegionsPointer + 1) And Regions(0, RegionsPointer) <= Regions(1, RegionsPointer + 1)
    ' yoverlap (next to current)
    ' Regions(3, RegionsPointer) >= Regions(2, RegionsPointer + 1) And Regions(2, RegionsPointer) <= Regions(3, RegionsPointer + 1)

    RegionMatchFound = 0
    RegionEnd = RegionsNumber - RegionsPointer

    For RegionsMatchPointer = 1 To RegionEnd
If Regions(1, RegionsPointer + RegionsMatchPointer) >= Regions(0, RegionsPointer) And Regions(0, RegionsPointer + RegionsMatchPointer) <= Regions(1, RegionsPointer) And Regions(2, RegionsPointer) >= Regions(3, RegionsPointer + RegionsMatchPointer) Then
   'do a merge - exit for
   If Regions(0, RegionsPointer) <= Regions(0, RegionsPointer + RegionsMatchPointer) Then
      Regions(0, RegionsPointer + RegionsMatchPointer) = Regions(0, RegionsPointer)
   End If
   If Regions(1, RegionsPointer) >= Regions(1, RegionsPointer + RegionsMatchPointer) Then
      Regions(1, RegionsPointer + RegionsMatchPointer) = Regions(1, RegionsPointer)
   End If
   If Regions(2, RegionsPointer) <= Regions(2, RegionsPointer + RegionsMatchPointer) Then
      Regions(2, RegionsPointer + RegionsMatchPointer) = Regions(2, RegionsPointer)
   End If
   If Regions(3, RegionsPointer) >= Regions(3, RegionsPointer + RegionsMatchPointer) Then
      Regions(3, RegionsPointer + RegionsMatchPointer) = Regions(3, RegionsPointer)
   End If
   RegionMatchFound = 1
   Exit For
   ElseIf Regions(1, RegionsPointer) >= Regions(0, RegionsPointer + RegionsMatchPointer) And Regions(0, RegionsPointer) <= Regions(1, RegionsPointer + RegionsMatchPointer) And Regions(3, RegionsPointer) >= Regions(2, RegionsPointer + RegionsMatchPointer) And Regions(2, RegionsPointer + RegionsMatchPointer) <= Regions(3, RegionsPointer + RegionsMatchPointer) Then
      'do a merge - exit for
      If Regions(0, RegionsPointer) <= Regions(0, RegionsPointer + RegionsMatchPointer) Then
         Regions(0, RegionsPointer + RegionsMatchPointer) = Regions(0, RegionsPointer)
      End If
      If Regions(1, RegionsPointer) >= Regions(1, RegionsPointer + RegionsMatchPointer) Then
         Regions(1, RegionsPointer + RegionsMatchPointer) = Regions(1, RegionsPointer)
      End If
If Regions(2, RegionsPointer) <= Regions(2, RegionsPointer + RegionsMatchPointer) Then
    Regions(2, RegionsPointer + RegionsMatchPointer) = Regions(2, RegionsPointer)
End If
If Regions(3, RegionsPointer) >= Regions(3, RegionsPointer + RegionsMatchPointer) Then
    Regions(3, RegionsPointer + RegionsMatchPointer) = Regions(3, RegionsPointer)
End If
RegionMatchFound = 1
Exit For
End If

'if no match found, then this region must be an object
If RegionMatchFound = 0 Then
    For i = 0 To 3
        Objects(i, ObjectsNumber) = Regions(i, RegionsPointer)
    Next i
    ObjectsNumber += 1
End If

'now reduce the objects by the region minimum size factor
For ObjectsPointer = 0 To ObjectsNumber - 1
    Objects(0, ObjectsPointer) = Objects(0, ObjectsPointer) + RegionMinimumDistance
    Objects(1, ObjectsPointer) = Objects(1, ObjectsPointer) - RegionMinimumDistance
    Objects(2, ObjectsPointer) = Objects(2, ObjectsPointer) + RegionMinimumDistance
    Objects(3, ObjectsPointer) = Objects(3, ObjectsPointer) - RegionMinimumDistance
    If Objects(0, ObjectsPointer) < 0 Then
        Objects(0, ObjectsPointer) = 0
End If
If Objects(1, ObjectsPointer) > PixelsPerRow - 1 Then
    Objects(1, ObjectsPointer) = PixelsPerRow - 1
End If

If Objects(2, ObjectsPointer) < 0 Then
    Objects(2, ObjectsPointer) = 0
End If

If Objects(3, ObjectsPointer) > NumberOfRows - 1 Then
    Objects(3, ObjectsPointer) = NumberOfRows - 1
End If

Next

'finally, get rid of the small objects

'UnlockBitmap(regionbmap)

PictureBox5.Image = bmap

'now draw onto the image for each object

Dim b As Bitmap
Dim g As Graphics

b = New Bitmap(PictureBox5.Image)
g = Graphics.FromImage(b)

g.DrawLine(Pens.Red, Xstart, Ystart, Xend, Yend)

For i = 0 To ObjectsNumber - 1
    g.DrawLine(Pen, Objects(0, i), Objects(2, i), Objects(1, i), Objects(2, i))
g.DrawLine(Pen, Objects(0, i), Objects(2, i), Objects(0, i), Objects(3, i))
g.DrawLine(Pen, Objects(1, i), Objects(2, i), Objects(1, i), Objects(3, i))
g.DrawLine(Pen, Objects(0, i), Objects(3, i), Objects(1, i), Objects(3, i))
'g.DrawString(i, drawFont, Brushes.Red, Objects(0, i), Objects(2, i))

Next i
PictureBox5.Image = b

'end object growth
------------------------------------------------

'------------------------------------------------

'Object Tracking
'------------------------------------------------

'the following will attempt to track an object based simply on
'size and position

'1st create an array for the new object info

'format - size, xcentre, ycentre

Dim CurrentObjects(6, Length) As Integer     'size, xcentre, ycentre, matchfound

For ObjectsPointer = 0 To ObjectsNumber - 1

    'find size
    CurrentObjects(0, ObjectsPointer) = (Objects(1, ObjectsPointer) - Objects(0, ObjectsPointer)) * (Objects(3, ObjectsPointer) - Objects(2, ObjectsPointer))

    'find xcentre point
    CurrentObjects(1, ObjectsPointer) = Math.Ceiling((Objects(1, ObjectsPointer) - Objects(0, ObjectsPointer)) / 2) + Objects(0, ObjectsPointer)

    'find ycentre point
    CurrentObjects(2, ObjectsPointer) = Math.Ceiling((Objects(3, ObjectsPointer) - Objects(2, ObjectsPointer)) / 2) + Objects(2, ObjectsPointer)
Next

`Dim PossibleObjectsNumberTemp As Integer = 0` 'the number of current possible and matched objects

`MatchFound = 0` 'used to exit the for loop

`Dim MatchSize As Integer = 20000` 'used to compare size

`Dim MatchPosition As Integer = 200` 'used to compare position

'these are any new objects which don't have a match with the confirmed objects array

'clean out possible object matches

For PossibleObjectsPointer = 0 To PossibleObjectsNumber - 1
    PossibleObjects(4, PossibleObjectsPointer) = 0
Next

'1st compare all possible objects with the current objects

For ObjectsPointer = 0 To ObjectsNumber - 1
    For PossibleObjectsPointer = 0 To PossibleObjectsNumber - 1
        If (CurrentObjects(0, ObjectsPointer) + MatchSize) > PossibleObjects(0, PossibleObjectsPointer) And (CurrentObjects(0, ObjectsPointer) - MatchSize) < PossibleObjects(0, PossibleObjectsPointer) And (CurrentObjects(1, ObjectsPointer) + MatchPosition) > PossibleObjects(1, PossibleObjectsPointer) And (CurrentObjects(1, ObjectsPointer) - MatchPosition) < PossibleObjects(1, PossibleObjectsPointer) And (CurrentObjects(2, ObjectsPointer) + MatchPosition) > PossibleObjects(2, PossibleObjectsPointer) And (CurrentObjects(2, ObjectsPointer) - MatchPosition) < PossibleObjects(2, PossibleObjectsPointer)
            Then
                'update the possible objects with the current object
                For i = 0 To 2
                    PossibleObjects(i, PossibleObjectsPointer) = CurrentObjects(i, ObjectsPointer)
                Next
                PossibleObjects(3, PossibleObjectsPointer) += 1 'how many times has this object been matched
                PossibleObjects(4, PossibleObjectsPointer) = 1 'a match has been found for this possible object
USB Camera Pedestrian Counting

CurrentObjects(3, ObjectsPointer) = 1  'a match has been found, do not add this to the possible list
End If
Next
Next

'now clean up the possible objects - delete possibles with no matches
For PossibleObjectsPointer = 0 To PossibleObjectsNumber
  If PossibleObjects(4, PossibleObjectsPointer) = 0 Then
    PossibleObjects(3, PossibleObjectsPointer) -= 1
    If PossibleObjects(3, PossibleObjectsPointer) < 1 Then
      'mark this entry for deletion
      PossibleObjects(5, PossibleObjectsPointer) = 1
    End If
  End If
Next

'do the deletions, update pointers and the possible objects array
For PossibleObjectsPointer = 0 To PossibleObjectsNumber - 1
  If PossibleObjects(5, PossibleObjectsPointer) = 0 Then
    For i = 0 To 6
      PossibleObjectsTemp(i, PossibleObjectsPointer) = PossibleObjects(i, PossibleObjectsPointer)
    Next
    PossibleObjectsNumberTemp += 1
  End If
Next
PossibleObjectsNumber = PossibleObjectsNumberTemp

'clean the possible array before the copy operation
For PossibleObjectsPointer = 0 To PossibleObjectsNumber + 10
  For i = 0 To 6
    PossibleObjects(i, PossibleObjectsPointer) = 0
  Next
Next
USB Camera Pedestrian Counting

\[
\text{For } \text{PossibleObjectsPointer} = 0 \text{ To } \text{PossibleObjectsNumberTemp} - 1 \\
\hspace{1em} \text{For } i = 0 \text{ To } 6 \\
\hspace{2em} \text{PossibleObjects}(i, \text{PossibleObjectsPointer}) = \text{PossibleObjectsTemp}(i, \\
\hspace{2em} \text{PossibleObjectsPointer}) \\
\hspace{1em} \text{Next} \\
\text{Next}
\]

'now any objects from the current array which didn't have a match should be copied to the
possible array

\[
\text{For } \text{ObjectsPointer} = 0 \text{ To } \text{ObjectsNumber} - 1 \\
\hspace{1em} \text{If } \text{CurrentObjects}(3, \text{ObjectsPointer}) = 0 \text{ Then} \\
\hspace{2em} \text{For } i = 0 \text{ To } 5 \\
\hspace{3em} \text{PossibleObjects}(i, \text{PossibleObjectsNumber}) = \text{CurrentObjects}(i, \text{ObjectsPointer}) \\
\hspace{2em} \text{Next} \\
\hspace{1em} \text{PossibleObjectsNumber} += 1 \\
\hspace{1em} \text{End If} \\
\text{Next}
\]

'next clean up the duplicate entries in the possible array

\[
\text{Dim } \text{SearchLength } \text{As Integer} = 0 \\
\text{Dim } \text{SearchPointer } \text{As Integer} = 0 \\
\text{SearchLength} = \text{PossibleObjectsNumber}
\]

\[
\text{For } \text{PossibleObjectsPointer} = 0 \text{ To } \text{PossibleObjectsNumber} - 1 \\
\hspace{1em} \text{For } \text{SearchPointer} = 1 \text{ To } \text{SearchLength} \\
\hspace{2em} \text{If } \text{PossibleObjects}(0, \text{PossibleObjectsPointer}) = \text{PossibleObjects}(0, \\
\hspace{2em} \text{PossibleObjectsPointer} + \text{SearchPointer}) \text{ And } \text{PossibleObjects}(1, \text{PossibleObjectsPointer}) = \text{PossibleObjects}(1, \\
\hspace{2em} \text{PossibleObjectsPointer} + \text{SearchPointer}) \text{ Then} \\
\hspace{3em} \text{If } \text{PossibleObjects}(3, \text{PossibleObjectsPointer}) < \text{PossibleObjects}(3, \\
\hspace{3em} \text{PossibleObjectsPointer} + \text{SearchPointer}) \text{ Then} \\
\hspace{4em} \text{PossibleObjects}(3, \text{PossibleObjectsPointer}) = \text{PossibleObjects}(3, \\
\hspace{4em} \text{PossibleObjectsPointer} + \text{SearchPointer}) \\
\hspace{2em} \text{End If}
\hspace{1em} \text{End If}
\hspace{1em} \text{Next} \\
\text{Next}
\]

119
For i = 0 To 5
    PossibleObjects(i, PossibleObjectsPointer + SearchPointer) = 0
Next i
End If
Next
SearchLength -= 1
Next

' prep drawing for next section

' now draw onto the image for each object

'Dim b As Bitmap
'Dim g As Graphics

b = New Bitmap(PictureBox5.Image)
g = Graphics.FromImage(b)

' g.DrawLine(Pens.Red, Xstart, Ystart, Xend, Yend)
' For i = 0 To ObjectsNumber - 1                  ' g.DrawLine(Pen, Objects(0, i), Objects(2, i), Objects(1, i), Objects(2, i))
' g.DrawLine(Pen, Objects(0, i), Objects(2, i), Objects(0, i), Objects(3, i))
' g.DrawLine(Pen, Objects(1, i), Objects(2, i), Objects(1, i), Objects(3, i))
' g.DrawLine(Pen, Objects(0, i), Objects(3, i), Objects(1, i), Objects(3, i))
' g.DrawString(i, drawFont, Brushes.Red, Objects(0, i), Objects(2, i))

' Next i

' now lets find the possible objects with a high matchcounter and draw on these with their entry position

Dim MatchThreshold As Integer = 20
Dim MatchLockOn As Integer = 20

For PossibleObjectsPointer = 0 To PossibleObjectsNumber - 1
If PossibleObjects(3, PossibleObjectsPointer) > MatchLockOn And ObjectsMatchStatus(0, PossibleObjectsPointer) = 0 Then
    ObjectsMatchStatus(0, PossibleObjectsPointer) = 1
    'draw on object
    g.DrawString(PossibleObjectsPointer + 1, drawFont, Brushes.Green, PossibleObjects(1, PossibleObjectsPointer), PossibleObjects(2, PossibleObjectsPointer))
    NumberOfPedestrians += 1
    Label1.Text = "Number Of Pedestrians " + CStr(NumberOfPedestrians)
    PossibleObjects(3, PossibleObjectsPointer) = MatchThreshold + MatchLockOn
End If

If PossibleObjects(3, PossibleObjectsPointer) > MatchThreshold And ObjectsMatchStatus(0, PossibleObjectsPointer) = 1 Then
    'draw on object
    g.DrawString(PossibleObjectsPointer + 1, drawFont, Brushes.Red, PossibleObjects(1, PossibleObjectsPointer), PossibleObjects(2, PossibleObjectsPointer))
    If PossibleObjects(3, PossibleObjectsPointer) > MatchThreshold + MatchLockOn Then
        PossibleObjects(3, PossibleObjectsPointer) = MatchThreshold + MatchLockOn
    End If
End If

If PossibleObjects(3, PossibleObjectsPointer) < MatchThreshold And ObjectsMatchStatus(0, PossibleObjectsPointer) = 1 Then
    'track has been lost
    PossibleObjects(3, PossibleObjectsPointer) = 0
    ObjectsMatchStatus(0, PossibleObjectsPointer) = 0
End If

Next
PictureBox5.Image = b
g.Dispose()

End If

'display the current frame number before repeating the loop
FRAME_RATE_COUNTER += 1
USB Camera Pedestrian Counting

' Label1.Text = "Frame Number = " + CStr(FRAME_RATE_COUNTER)

' allow the user form to run for 1 cycle - this means button etc can be used
Application.DoEvents()

End While

End Sub

End Sub

Private Sub btnLockBits_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    RUN_SYSTEM = 1
    ProcessImage()

End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    RUN_SYSTEM = 0
    FRAME_RATE_COUNTER = 0

End Sub

Private Sub Label2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)

End Sub

Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)

End Sub

Private Sub PictureBox3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PictureBox3.Click

End Sub
Private Sub HScrollBar1_Scroll(ByVal sender As System.Object, ByVal e As System.Windows.Forms.ScrollEventArgs) Handles HScrollBar1.Scroll
    ChangeThreshold = HScrollBar1.Value
    Label5.Text = HScrollBar1.Value.ToString
End Sub

Private Sub HScrollBar4_Scroll(ByVal sender As System.Object, ByVal e As System.Windows.Forms.ScrollEventArgs) Handles HScrollBar4.Scroll
    GhostFilter = HScrollBar4.Value
End Sub

Private Sub HScrollBar3_Scroll(ByVal sender As System.Object, ByVal e As System.Windows.Forms.ScrollEventArgs) Handles HScrollBar3.Scroll
    RegionMinSize = HScrollBar3.Value
End Sub

Private Sub HScrollBar2_Scroll(ByVal sender As System.Object, ByVal e As System.Windows.Forms.ScrollEventArgs) Handles HScrollBar2.Scroll
    RegionMinimumDistance = HScrollBar2.Value
    Label17.Text = HScrollBar2.Value.ToString
End Sub

Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label1.Click
    Label1.Text = "Frame Number = " + CStr(FRAME_RATE_COUNTER)
End Sub

Private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RadioButton1.CheckedChanged
    If RadioButton1.Checked = True Then
        If RUN_SYSTEM = 1 Then
            RUN_SYSTEM = 0
        End If
    End If
End Sub
USB Camera Pedestrian Counting

Image_Size = 1
FRAME_RATE_COUNTER = 0
RUN_SYSTEM = 1
ProcessImage()

Else : Image_Size = 1
End If
End If

If RadioButton2.Checked = True Then
  If RUN_SYSTEM = 1 Then
    RUN_SYSTEM = 0
    Image_Size = 2
    FRAME_RATE_COUNTER = 0
    RUN_SYSTEM = 1
    ProcessImage()

    Else : Image_Size = 2
    End If
  End If
End If

If RadioButton3.Checked = True Then
  If RUN_SYSTEM = 1 Then
    RUN_SYSTEM = 0
    Image_Size = 4
    FRAME_RATE_COUNTER = 0
    RUN_SYSTEM = 1
    ProcessImage()
Else : Image_Size = 4
End If
End If

Private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RadioButton2.CheckedChanged
    If RadioButton1.Checked = True Then
        If RUN_SYSTEM = 1 Then
            RUN_SYSTEM = 0
            Image_Size = 1
            FRAME_RATE_COUNTER = 0
            RUN_SYSTEM = 1
            ProcessImage()
        Else : Image_Size = 1
        End If
    End If
End If

If RadioButton2.Checked = True Then
    If RUN_SYSTEM = 1 Then
        RUN_SYSTEM = 0
        Image_Size = 2
        FRAME_RATE_COUNTER = 0
        RUN_SYSTEM = 1
        ProcessImage()
Else : Image_Size = 2
End If
End If

If RadioButton3.Checked = True Then

If RUN_SYSTEM = 1 Then

RUN_SYSTEM = 0
Image_Size = 4
FRAME_RATE_COUNTER = 0
RUN_SYSTEM = 1
ProcessImage()

Else : Image_Size = 4
End If
End If
End Sub
End Class

Imports System.Drawing.Imaging
Imports System.Runtime.InteropServices

Module LockBitmapStuff

Public g_RowSizeBytes As Integer
Public g_PtrBytes() As Byte
Private m_BitmapData As BitmapData

' Lock the bitmap's data.
Public Sub LockBitmap(ByVal bm As Bitmap)
' Lock the bitmap data.
Dim bounds As Rectangle = New Rectangle( _
  0, 0, bm.Width, bm.Height)
m_BitmapData = bm.LockBits(bounds, _
        Imaging.ImageLockMode.ReadWrite, _
        Imaging.PixelFormat.Format24bppRgb)
g_RowSizeBytes = m_BitmapData.Stride

' Allocate room for the data.
Dim total_size As Integer = m_BitmapData.Stride * m_BitmapData.Height - 1
ReDim g_PixBytes(total_size)

' Copy the data into the g_PixBytes array.
Marshal.Copy(m_BitmapData.Scan0, g_PixBytes, _
    0, total_size)

End Sub

Public Sub UnlockBitmap(ByVal bm As Bitmap)
' Copy the data back into the bitmap.
Dim total_size As Integer = m_BitmapData.Stride * m_BitmapData.Height
Marshal.Copy(g_PixBytes, 0, _
    m_BitmapData.Scan0, total_size)

' Unlock the bitmap.
bm.UnlockBits(m_BitmapData)

' Release resources.
g_PixBytes = Nothing
m_BitmapData = Nothing
End Sub
End Module
7.12 Practical Deployment

In order to deploy this type of system some practical considerations need to be made. The issues are:

1. The USB standard has a cable length limit of 5m, or 30m when using active repeaters. This would make it difficult for deployment as the computer would have to be close to the camera.

2. An IP camera could be used however this would increase the cost of the system by at least $1000. If an IP camera was used however a wireless Ethernet bridge could then be used.

3. If mounting the system outdoors, a pole mounted IP65 rated enclosure would be required.

4. If a laptop was stored inside of the enclosure a general purpose outlet supplying 240VAC would also be required. IP rated USB cameras are available and this could be mounted under the enclosure.

5. Greater driver control is necessary and the software needs to be able to tune its parameters to suit a new environment. These two issues are discussed in the results section of this paper.
System costs for the physical deployment shows in figure 40 would be:

1. Pole Installation, Labour and Supply $2000
2. Enclosure $500
3. Laptop $1000
4. Camera $200
5. Electrical Labour, 10 hours * 2 people $1100

Total System Cost Estimate: $4800

FIGURE 40 - FIELD DEPLOYMENT
7.13 Linux Deployment

Laptops with a fully working Windows 7 OS are now available for less than $500. The software if optimized could be made to run with lower processing capabilities than the platform used for development would be suitable for low cost laptops using Duo cores or an equivalent. The author’s experience with Linux during the USQ Electrical and Electronic practice involved the use of virtual machines running Linux. The main issue encountered with Linux is the higher levels of computing expertise required by the end user. If deploying the software, it would have to be user-friendly and the high majority of computer users are familiar with Windows conventions and Linux. Another issue is the lack of drivers available for hardware. In order to get a working product, it is likely a driver would have to be created for the Linux platform. This would limit the cameras which could be used for the application. If the aim was to develop a user friendly commercially viable application which could be setup by the end-user and use a wide range of off the shelf Webcams, Linux is not recommended.
7.14 Other Applications

The most obvious application for this type of system is vehicle detection, counting and control. Being able to optimize traffic on/off times or plan upgrades are two possible practical applications. More futuristic is the use of fully automated transport where the vehicle drives itself and uses vision system technologies. Other applications would be the detection of anti-social behavior of people by monitoring for violent actions of people.

To adapt this system to vehicle counting characteristics which are unique to vehicles would have to be described and tracked. Active shape tracking or model fitting would be required to ensure the system can differentiate between vehicles or non-vehicles.

The methods used here show great promise for being the basis of a fully functioning pedestrian tracker. As there already exist working tracking systems built on this technology, it is apparent that the method is proven. This is an exciting field which is still in its infancy. Vision systems hold great promise for increasing the levels of automation within society.
8 CONCLUSIONS

8.1 Result Project Achievements

1. Research and identify the most appropriate programming language for the project and develop a working knowledge of the chosen language.

This objective has been achieved, although the search was not exhaustive and did not consider every language. The selection was primarily based on what tools were freely available to the author, which language could be learnt in the timeframe given, and which language would most likely be used by the author in future works. A basic working knowledge of the language was developed, however broad programming concepts were not understood and as such, issues encountered with multithreading and function creation could not be overcome. An installable application which ran in real time, accessed the USB camera and gave the user the ability to interact with the program was developed.

2. Research current theories and algorithms used in the field of vision systems, shape and pattern recognition and object tracking.

This objective was achieved and a classical tracking systems approach was taken towards the project. An overview of the various approaches to object tracking was
undertaken, however more classical methods were opted for. These classical methods are computationally inexpensive and with enough development of the tracking routines show good accuracy. Three projects were synthesized to provide a top down design for the project.

3. Design and write the software.

Approximately 70% of time was spent writing and debugging the software for the people tracking system. While conceptually the approach taken by the author was simple, the practical implementation in Visual Basic of the concepts was time consuming. In particular, the time taken to implement the region growing algorithm consumed the majority of the project build time due to numerous bugs. The final program is essentially a series of filters which run consecutively. The design and write objective was not completely achieved; the original design was altered significantly, the program could not cope with complex tracking scenarios, and the colour tracking was not implemented. A unique approach to object creation using the pixelator to overcome object segregation was trialled, but found to be redundant once the region growing was modified to include object growing.

4. Test the software and record the results.

The software test results are included with this project as images, attached videos and qualitative statements only. No empirical method was developed to measure the
program’s performance apart from speed measurements. The video results however are sufficient to demonstrate the limitations of the method chosen by the author and these results were used by the author to identify the flaws in the current system.

5. If the written program is successful in a basic test environment, trial the system in more difficult conditions, identify flaws, and improve the program resiliency to changes in camera perspective and lighting.

The program improvements after testing in the basic office environment using a single test subject involved tuning system parameters which lead to improved performance of the software. It became obvious during outdoor trials that the tracking routines developed is insufficient for any scenario involving multiple subjects and shadows. Lighting changes and perspective changes were not addressed as these were not the fundamental issues affecting the program performance. The primary issues encountered were the poor quality of the motion image, the effects of shadows and the lack of development in the tracking routines.

6. Discuss system costs in terms of computer hardware and mounting enclosure required for practical installations.

This objective was conceptually considered and a practical installation design is given. Further detailed design would be required and this would involve the
dimensioning of the mounting pole and enclosure, sourcing materials and supervising construction. Cost comparisons to other methods was not made and is necessary to comment on the cost effectiveness of the USB camera approach.

7. Consider developing the system for linux to lower costs using a cross platform language.

This is option recommended due to the lack of hardware driver support and higher levels of computing expertise require by both the end-user and the developer.

8. Consider using the software for vehicular traffic and the changes to the software required.

This was briefly considered and the software in its current form would have more success tracking traffic than tracking people assuming the perspective of the camera overlooking the highway was ideal. Additional modules which differentiate between object types would be required.

9. Consider using the software for traffic light control enhancement.
USB Camera Pedestrian Counting

This objective was not addressed to any significant extent except to note the possibility of optimising traffic flow by an ongoing analysis of traffic patterns.

10. Identify other applications for this type of system.

This objective was not directly addressed and only the possibility of applying these technologies generally to process control and automation was considered.
8.2 Recommendations For Future Work

8.2.1 Difference Image Improvements

The difference image requires filtering in order to improve the results of the system. A simple noise filter which averages the image could be applied to remove some of the speckled noise which was occurring in the difference image.

Shadow removal techniques could be developed to help overcome the false positives occurring due to the multiple shadows which tend to occur in confined spaces and when the sun is not directly overhead. The shadow removal techniques developed below relies on a stereo image and the codebook method (Amitpal5624, 2008).

FIGURE 41 – SHADOW DETECTION – SOURCE: (AMITPAL5624, 2010)
8.2.2 Image Size Reduction

The software developed gave the user the option of changing the image size. By reducing the size of the image from 640 x 480 to 160 x 120 the number of pixels which needed to be processed was reduced to 1/16\textsuperscript{th} of the original number. This lead to some speed increases. The speed increase was approximately 7 fps. The results of the program were not significantly different during these image resizes. Various parameters needed to be adjusted for the smaller image size, however the rest of the routines ran successfully using the lower resolution. It is possible that a very small image could be used and that this would significantly increase the speed, or decrease the computing resources needed for a tracking system.
8.2.3 Occlusion Handling Routines

One of the main issues encountered was the overlap of objects. This caused object tracking loss in all tests. No routines were created to overcome occlusion scenarios. Methods to overcome this and other tracking loss scenarios are discussed in the section entitled additional modules.

8.2.4 Camera Control

The USB Logitech webcam used had multiple options for webcam control. In order to develop a working system it would be necessary to control the driver to a greater extent. It would also be necessary to change the camera settings based on the environment. Several variables within the program were adjusted to suit the office environment the software was developed in. In an outdoor environment as lighting changes it would be necessary to updates these variables and the camera lighting or gain control.
8.2.5 Additional Tracking Modules

The following modules need to be added to the software in order to make the software a working people tracker.

1. A velocity and position estimator for the object tracking routine.
2. Active shape fitting that identifies human shaped objects and only tracks these.
3. An omega (head) detector.
4. Region splitting to overcome occlusion and tracking loss.
5. Raw image and difference image filtering to improve the input to the motion detection section of the program.
6. Colour tracking routines to improve the performance of the system.

8.2.6 Software Improvements

A fixed frame rate needs to be used by the software. With a variable framerate the object tracking parameters such as size comparison thresholds and position threshold need to be varied to account for the difference in position transitions over each frame. For example, with a slow frame rate the movement between frames is greater. With a high frame the movement is smaller. The frame rate currently varies depending on the scene composition and the amount of noise.
Multithreading needs to be added to improve the speed of the system and to make available more CPU cores so that additional processing routines can occur.

Translation to a faster language should be performed. The C language is recommended to gain a speed increase for IO and greater hardware control.

Greater modularisation and code quality control is necessary to improve the software’s maintainability.

The program should sleep when there is no movement in a scene to conserve power. A basic motion detection routine could be written which compares the current frame to the previous frame. When the difference between the frames exceeds a threshold value, the main routine should be started and run.

Programming concepts such as object orientated, component driven etc were not fully grasped by the author throughout the project. The ability to create classes, or interpret user documentation of various functions or libraries was made more difficult due to the lack of this basic knowledge. Obtaining a stronger foundation in the basics of programming would be necessary to improve the speed and functionality of the software.
8.2.7 Future Research Topics

The breadth of the topic selected meant that improvements to existing research and techniques could not be achieved. This research project and the final application is essentially a synthesis of techniques which have been used since the 1990’s. The scope of future research should be more refined to allow for a greater contribution to the research community. Some recommendations for future titles would be Motion Detection Technique Comparisons, Mono vs Stereo Shadow Removal Techniques, or Pixel Grouping Techniques Speed Comparison. A comprehensive summary of current approaches to people tracking and their relative strengths and weaknesses would be beneficial to system designers and researchers.

8.2.8 Implement The Reading People Tracker

Source code is available for the Reading People Tracker at http://www.siebel-research.de/people_tracking/reading_people_tracker/. This system could be compiled and then re-engineered to develop a working people tracker. This would be a suitable project for a computer engineering student with a working knowledge of C++. The “research” value of this however would be questionable as it unlikely any new methods for vision systems would be pioneered.
9 LIST OF REFERENCES


19. Li, M et al, “Rapid And Robust Human Detection And Tracking Based On Omega-Shape Features”, IEEE, Image Processing (ICIP), 2009 16th IEEE International Conference on, 2009, pp.2545-2548


21. Thirumuruganathan, S, “Introduction To Mean Shift Algorithm”, Wordpress Weblog,


25. Amitpal5624, “Shadow Detection”, YouTube, 
http://www.youtube.com/watch?v=zSL5WwY0rE, Date Accessed: 19/10/2010
10 APPENDIX A – PROJECT SPECIFICATION

For: Jeremy Bruce Duncan
Topic: Pedestrian Traffic Monitoring using Machine Vision
Supervisor: Professor John Billingsley
Project Aim: To develop software for counting people using a USB camera as the sensing device.

Programme:

1. Research and identify the most appropriate programming language for the project and develop a working knowledge of the chosen language.
2. Research current theories and algorithms used in the field of vision systems, shape and pattern recognition and object tracking.
3. Design and write the software.
4. Test the software and record the results.
5. If the written program is successful in a basic test environment, trial the system in more difficult conditions, identify flaws, and improve the program resiliency to changes in camera perspective and lighting.

As time permits:

1. Discuss system costs in terms of computer hardware and mounting enclosure required for practical installations.
2. Consider developing the system for linux to lower costs using a cross platform language.
3. Consider using the software for vehicular traffic and the changes to the software required.
4. Consider using the software for traffic light control enhancement.
5. Identify other applications for this type of system.

AGREED: ____________________________ (Student) Date:

AGREED: ____________________________ (Supervisor) Date:

Examiner/Co-Examiner: ____________________________
11 APPENDIX B – POWER POINT PRESENTATION
USB Camera Pedestrian Counting

Language Selection

- Matlab Tests Too Slow
- Free Visual Studio 2008 Available
- C Language Longer Learning Curve
- VB – Fast Enough and Easy to Learn
- AB PLC, Citect Scada, Excel

Language Selection – BenchMarks

Research – Current Systems

- Non-Adaptive Artificial Environments
- Kalman Filters for Background and Color Tracking
- Mean Shift Algorithm
- Reading People Tracker

Language Selection – Image Acquisition

- 640x480, 24bpp RGB BMP
- AV/Codec Class
- Lock/Unlock Bits Method
- 30fps
- Hardware, Logitech Cam, 2.63GHz Quad Core, 64bit OS
The Motion Detector

- Background Image Generation Followed By Differencing
- Background Generated Using Median Filter
- Approximate Median Filter Used

\[ \text{Pixel}_L = \text{Average}(\text{Pixel}_{L-1}, \text{Pixel}_L, \ldots, \text{Pixel}_{L+n}) \]
The Difference Image

Difference Image
For all n

if |Pixel|<current> − Pixel|<median>| > Motion_Threshold Then
SetWhite

Trial Greyscale

Constructing Objects – Advanced Region Growing Algorithm

Constructing Objects

LINE BLOBS > REGIONS > OBJECTS
Scan Row A
Scan Row B
Merge A To B
New Regions?
Continue

Constructing Objects – Build Regions

Constructing Objects – Region Collision Logic
USB Camera Pedestrian Counting

Constructing Objects – Build Regions

Objects – Final Result

Software Engineering

- Speed – fps, resize, C
- Multithreading
- Calls to C/C++ Dlls
- Exception Handling
- Detect Motion To Start Filter
- Efficient Nesting

Basic Tracking

- Collect Objects
- Find Size and Centre of Object
- Compare all Current Objects to All Past Objects
- Assign ID
- Object Not Found for >5 frames
- Increment People Counter

Basic Tracking

Outdoor Results

- Region Min Size Causing Object Segregation
- Shadows Causing Object Fusion
- Debug, Redesign, Tracking Technique
- How to Compensate for Object Distance and Fixed Parameters Used For Matching
- New Environment, New Constants
Tracking Improvements

- Improve Difference Image
- Velocity Estimates
- Future Position Estimates, 1st order
- Region Splitting/Merging
- Head Detector
- Shape Fitting
- RGB Histogram
- Currently an Object Tracker – Not People

Practical Deployment

- USB 2.0 Cable 5m Limit
- USB Cables With Active Repeaters
- IP Camera, Copper to Optic/Wireless, High Cost, 27MB/s
- Self-Adjusting Parameters
- Greater Camera Control Required
- Environmental Issues
- Equipment IP Ratings

Future Research

- Human Vision
- Implement C Source of Reading System
- Driver Creation
- Difference Image Filters
- Shadow Removal Techniques
- Frequency Analysis Techniques
12 Appendix C - YouTube Postings

Vision system research can be assisted by browsing YouTube for relevant videos. It gives the researcher a multitude of videos which show different techniques in use. These videos allow for rapid understanding of what results can be achieved from a vision systems method. The author of this paper has posted videos for this project onto YouTube to show the results of this project.

The following titles can be searched and played on YouTube:

“Difference Image” - [http://www.youtube.com/watch?v=xzedig8rwJ0](http://www.youtube.com/watch?v=xzedig8rwJ0)

Description: The Difference Image of the Median Filtered Background Image and the current foreground image.

“Median Filtered Background Image” -
[http://www.youtube.com/watch?v=Lfl2g3EUVxU](http://www.youtube.com/watch?v=Lfl2g3EUVxU)

Description: The Median Filtered Background Image Used to Create a Difference Image. The median filter used is the approximate median filter which is faster. Notice how when I stop moving my hand it becomes part of the background image.
USB Camera Pedestrian Counting

“Region Growing for Object Tracking” -
http://www.youtube.com/watch?v=wBZB2K7rvJc

Description: Shows the advanced region growing method with some modifications. The rectangle shows where the program thinks motion is occurring and groups areas it thinks is part of 1 moving object. A simple test environment is shown. It groups adjacent pixels into blobs, groups of blobs into regions and groups of regions into objects.

Simple People Tracker - http://www.youtube.com/watch?v=1fGJp2JUUvM

The tracker counts people in a very simple environment only. This pedestrian counter will not handle occlusion and gets confused when too many shadows are in the scene. People tracking software using difference image, median filtered background image, region growing and simple object tracking routines.