MIDLANDS STATE UNIVERSITY

FACULTY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF APPLIED PHYSICS AND TELECOMMUNICATIONS

"Internet Of Things (IoT) for Optimum Bus Infrastructure ".

By

Tapiwa Kadzimu

Reg Number R14717T

A DISSERTATION SUBMITTED TO THE FACULTY OF SOCIAL SCIENCES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE BACHELOR SCIENCE TELECOMMUNICATIONS HONOURS DEGREE

GWERU, ZIMBABWE

2018

DECLARATION v
APPROVAL vi
Dedication vii
Acknowledgements viii
ABSTRACT ix
List of Acronyms and Abbreviations x
CHAPTER 1 .... - 1 -
1.1 INTRODUCTION .... - 1 -
1.2 Aim and objectives of the Research .... - 1 -
1.3 Methodology .... - 2 -
1.4 List of resources Required .... - 2 -
1.4.1 Hardware requirement .... - 2 -
1.4.2 Software requirement .... - 3 -
1.5 Limitations .... - 3 -
References .... - 4 -
CHAPTER 2 .... - 5 -
2.LIERATURE REVIEW .... - 5 -
2.1 Introduction .... - 5 -
2.1.1Portability .... - 7 -
2.1.2 Image Filtering: .... - 8 -
2.1.3 Image Transformation .... - 8 -
2.1.4 Object Tracking: .... - 8 -
2.1.5 Feature Detection: .... - 8 -
2.2 OPENCV STRUCTURE AND CONTENT .... - 9 -
2.2.1 CX CORE .... - 9 -
2.2.2 IMGPROC .... - 9 -
2.2.4 HighGUI .... - 9 -
2.3 Image and Video Transforms .... - 11 -
2.3.1 Overview .... - 11 -
2.3.2 Convolution .... - 11 -
2.3.3 Scharr Filter .... - 12 -

2.3.4 Laplace ....- 12 -
2.4 People counting technologies used before ....- 13 -
2.4.1 Video Cameras ....- 13 -
2.4.2 Ultrasonic Sensors ....- 13 -
2.4.3 Infrared Sensor ....- 14 -
2.4.4 Infrared Motion Sensors ....- 14 -
2.5 Open CV Object Recognition ....- 14 -
2.5.1Object Recognition with Machine Learning Algorithms ....- 14 -
2.5.2 Recognition of Moving Objects through Background Subtraction Algorithms ....- 17 -
References ....- 21 -
CHAPTER 3 ....- 22 -
3.1 Current methods of people counting ....- 22 -
3.2 DESIGN AND IMPLEMENTATION ....- 24 -
3.4 Deep Learning using YOLO ....- 28 -
3.4.1 How YOLO works ....- 28 -
3.5 YOLO detection the objects in a given image ....- 29 -
3.7 Why use OpenCV for YOLO ? ....- 30 -
References ....- 31 -
CHAPTER 4 ....- 32 -
4. Results and Analysis ....- 32 -
4.1 1Introduction ....- 32 -
4.2 Functional Test Results ....- 32 -
4.2.1 People Counting ....- 32 -
4.2.2 Database Updating ....- 33 -
References. ....- 36 -
CHAPTER 5 ....- 37 -
5 CONCLUSION ....- 37 -
5.1 Introduction ....- 37 -
5.2 Real World Operation analysis ....- 37 -
5.3 Recommendation ....- 38 -
5.4 Summary and conclusion ....- 38 -

References - 39 -
APPENDICES - 40 -
Appendix A:Source Code - 40 -

DECLARATION

I Trevor Tapiwa Kadzimu hereby declare that I am the sole author of this dissertation. I authorize the Midlands State University to lend this dissertation to other individuals or institutions for the purpose of academic research.

Student: ………………………………………………………….

Date: …………………………………………………………….

APPROVAL

This dissertation entitled "Internet Of Things (IoT) for Optimum Bus Infrastructure " by Trevor meets the regulations governing the award of Bachelor of Science Honours Degree in Telecommunications by the Midlands State University, and is approved for its contribution to knowledge and literal presentation.

Supervisor: ………………………………………………………….

Date: ……………………………………………………………..

Dedication

I dedicate this special study piece to my family especially my wife Moreblessings Mafukidze and daughter Natalia Kadzimu for their enduring patience and faith in me during my studies .They tolerated and endured my perennial absence and still offered me support and hope .

Acknowledgements

First and foremost I would like to thank Lord God our Father for granting me the gift of life .Secondly i want to thank the entire Midlands State University Faculty of Science staff members for the guidance and mentorship they rendered throughout my studies .I would want to thank my supervisor Mr G Manjengwa for they special Academic attention and advice he gave me from the onset of my research project until the end . I also want to thank the Chairman of the Department of Applied Physics and Telecommunications Dr Nyamhere for the support he gave me through some of my difficulty times as a student .I would also want to thank my classmates for teamwork and study support they gave me through out the whole course

ABSTRACT

Rapid population growths or rural to urban migration and importation of small cheap private cars (mishikashika) has brought a lot of disarray in the public transport sector resulting n increased accidents ,traffic jams and inefficient transportation solutions in most major cities. The traditionally safer bus system is now underutilized and also ne sprouting illegal suburbs which are densely populated are being poorly serviced in terms of transport systems .this project seeks to bring an efficient solution for bus/transport allocation which is based on demand and fluctuations in demand during the whole day .it uses cameras and computer algorithms based on the open cv library suites to check bus stop status and alert the transport managers on which routes to prioritise as they can see the traffic profiles of different bus stops .computer learning ,human recognition and detection are the main concepts that are employed by this project to meet its objectives.

List of Acronyms and Abbreviations

- IOT -Internet of Things
- OpenCV -Open Computer V
- GSM -Global Sytems for Mobile communication
- ML -Machine L
- IMGPROC -Image Processer
- HMM -Hidden Markov. Model
- PIR -Passive Infrared
- EKF -Extended Kalman Filter
- HOG -(Histogram of Oriented Gradients
- SVM -Support Vector Machine, or
- SSD -Single Shot Detectors
- YOLO -You Only Look Once

CHAPTER 1

1.1 INTRODUCTION

In Zimbabwe, there is a need for an optimum efficient bus service infrastructure, smart proper bus transportation system. The increase in the population and the sprouting of new unplanned residential settlements has led to uncertain crowding at public bus stops. People wait for long hours and suddenly gather near stops whenever they see any bus coming nearby, even though it is not their destination bus. Also traditional large buses have lost ground to fast but often uncomfortable and dangerous small commuter omnibuses (kombis as they are locally called) due to their inefficiency and slowness. This has led to unnecessary crowding which can be solved by the use of this smart design of IOT based bus transport system. The small commuters have also congested the towns and have also caused many accidents

This system will not be giving real time feedback data to the waiting passengers about the next bus arrival times but it will count the number of people on bus stops and alert service providers who will in turn dispatch or reroute their buses to service the areas determined by this system. Depending on the data to be collected during the trial implementation the next bus arrival time for buses will be approximated. There is no certainty in the information regarding bus availability hence the system is forced to work under assumptions

So by using this system we can have information about the number of people on bus stops on the internet. What we need to do is just logon to the database and all the data related to certain routes can be seen in real time like the number of people on bus stops hence precise number of buses can be send through that route[1,2,3]

1.2 Aim and objectives of the Research

The research aim of this thesis is to create a camera application capable of detecting
the presence of humans on bus stops and organize acquired images in an online storage
service to achieve optimum bus service system through internet of things. The following research questions will be answered.

The objectives of this research being:

- ❖ Count the number of people at a bus-stop using cameras
- ❖ Store the number of people at the bus-stops in an online database
- ❖ Display the bus stop count on a Web based interface
- ❖ Generate an optimum road route to collect the passengers (requires internet)
- ❖ Display the optimum route on a map (requires internet)

1.3 Methodology

The aim of the project can be achieved by the use of an Internet of Things based system for optimum bus service infrastructure. This system consist of USB cameras for passengers counting, it uses a Raspberry Pi based control system for further data processing . This project also consists of GSM modem for remote monitoring and OpenCV Library.

The system will give a live count of people entering a bus top at any given time .The numbers are constantly updated sent to the central servers .Whenever the number of passengers reaches a specific number or goes above some predefined level, the information is stored in a data base and service providers will be alerted hence transport we will be provided on that specific route. The system will also be comprised of Google Street View API, Django Web Frame wake, Pythone IDE and MySQL to achieve the desired results.

1.4 List of resources Required

1.4.1 Hardware requirement

1) Raspberry Pi
2) Neo 6m GPS
3) USB camera (the higher the resolution the better) even HD camera)

(Wi-Fi camera’s will portray the system better)

4) Raspberry pi 4g Lte modern
5) Wifi router

1.4.2 Software requirement

1) Pythone IDE
2) Django Web Frame wake
3) Google Street View API
4) MySQL
5) OpenCV Library

1.5 Limitations

1) The destination of the passengers is not considered, the route generated is a pickup route and not a delivery route.
2) Bad weather, rain snow etc., my impair the camera's vision and miscount the number of people.
3) Requires internet which may not be available in rural setups

References

[1] K. Terada, D. Yoshida, S. Oe, and J. Yamaguchi, A method of counting the passing people by using the stereo images, International conference on image processing, 0-7803-5467-2016

[2] Haritaoglu and M. Flickner, Detection and tracking of shopping groups in stores, Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition , 0-7695- 1272-0,2016.

[3] Gary Conrad and Richard Johnsonbaugh, A real-time people counter, Proceedings of the 1994 ACM symposium on Applied computing, 0-89791-647-6 ,1994[ROS94] : M. Rossi and A. Bozzoli, Tracking and Counting Moving People, IEEE Proc. of Int. Conf. Image Processing, ,2014

CHAPTER 2

2.LIERATURE REVIEW

2.1 Introduction

Computer vision is a discipline, that deals with acquiring, processing as well as analyzing images[1]. Based on these functions, computer vision tries to extract as much useful information out of the images as possible, which can be used to make decisions. In this project, computer vision through opencv was used to count the number of people at a bust stop, extract the information including the gps coordinates and send to the server. Decisions on which route the bust would take will be based on that. A widespread theme has been the search for methods that could provide computers with human-like abilities in understanding images and deducting useful information out of it. Computer vision is frequently used in autonomous driving, object recognition as well as product quality management amongst others.

Computer vision has gone through a significant growth during the past decades and nowadays there are many libraries available for computer vision applications. The widespread usage of computer vision has been made possible by the combination of more capable algorithms, cheaper and more powerful hardware and better cameras[2]

OpenCV (Open Source Computer Vision Library) is a widely used computer vision and machine learning library mainly aimed at real-time applications. It was originally developed by Intel employees in a research center in Russia, but the project was taken over by a non-profit foundation in 2012[3]

According to [4] computer vision is the transformation of data from a still or video camera into either a decision or a new representation. All such transformations are done for achieving some particular goal, like in our case counting the number of people at a bus stop.

The input data may include some contextual information such as “the camera is mounted in a car” or “laser range finder indicates an object is 1 meter away”. The decision might be “there is a person in this scene” or “there are 14 tumor cells on this slide”. A new representation might mean turning a color image into a gray scale image or removing camera motion from an image sequence. Because we are such visual creatures, it is easy to be fooled into thinking that computer vision tasks are easy. How hard can it be to find, say, a car when you are staring at it in an image? Your initial intuitions can be quite misleading. The human brain divides the vision signal into many channels that stream different kinds of information into your brain. Your brain has an attention system that identifies, in a task-dependent way, important parts of an image to examine while suppressing examination of other areas. There is massive feedback in the visual stream that is, as yet, little understood. There are widespread associative inputs from muscle control sensors and all

of the other senses that allow the brain to draw on cross-associations made from years of living in the world. The feedback loops in the brain go back to all stages of processing including the hardware sensors themselves (the eyes), which mechanically control lighting via the iris and tune the reception on the surface of the retina. In a machine vision system, however, a computer receives a grid of numbers from the camera or from disk, and that's it. For the most part, there's no built-in pattern recognition, no automatic control of focus and aperture, no cross-associations with years of experience. For the most part, vision systems are still fairly naïve.

Given a two-dimensional (2D) view of a 3D world, there is no unique way to reconstruct the 3D signal. Formally, such an ill-posed problem has no unique or definitive solution. The same 2D image could represent any of an infinite combination of 3D scenes, even if the data were perfect. However, as already mentioned, the data is corrupted by noise and distortions. Such corruption stems from variations in the world (weather, lighting, reflections, movements), imperfections in the lens and mechanical setup, finite integration time on the sensor (motion blur), electrical noise in the sensor or other electronics, and compression artifacts after image capture.

Given these daunting challenges. In the design of a practical system, additional contextual knowledge can oft en be used to work around the limitations imposed on us by visual sensors.

Consider the example of a mobile robot that must find and pick up staplers in a building. The robot might use the facts that a desk is an object found inside offices and that staplers are mostly found on desks. This gives an implicit size reference; staplers must be able to fit on desks. It also helps to eliminate falsely "recognizing" staplers in impossible places for example on the ceiling or a window. The robot can safely ignore a 200-foot advertising blimp shaped like a stapler because the blimp lacks the prerequisite wood-grained background of a desk.

In contrast, with tasks such as image retrieval, all stapler images in a databasemay be of real staplers and so large sizes and other unusual configurations may have been implicitly precluded by the assumptions of those who took the photographs. That is, the photographer probably took pictures only of real, normal-sized staplers.

People also tend to center objects when taking pictures and tend to put them in characteristic orientations. Thus, there is often quite a bit of unintentional implicit information within photos taken by people. Contextual information can also be modeled explicitly with machine learning techniques. Hidden variables such as size, orientation to gravity, and so on can then be correlated with their values in a labeled training set. Alternatively, one may attempt to measure hidden bias variables by using additional sensors. The use of a laser range finder to measure depth allows us to accurately measure the size of an object.

The next problem facing computer vision is noise. We typically deal with noise by using statistical methods. For example, it may be impossible to detect an edge in an image merely by comparing a point to its immediate neighbors. But if we look at the statistics over a local region, edge detection becomes much easier. A real edge should appear as a string of such immediate neighbor responses

over a local region, each of whose orientation is consistent with its neighbors. It is also possible to compensate for noise by taking statistics over time. Still other techniques account for noise or distortions by building explicit models learned directly from the available data. For example, because lens distortions are well understood, one need only learn the parameters for a simple polynomial model in order to describe—and thus correct almost completely—such distortions.

Actions or decisions that computer vision attempts to make based on camera data are performed in the context of a specific purpose or task. We may want to remove noise or damage from an image so that our security system will issue an alert if someone tries to climb a fence or because we need a monitoring system that counts how many people cross through an area in an amusement park. Vision software for robots that wander through office buildings will employ different strategies than vision software for stationary security cameras because the two systems have significantly different contexts and objectives. As a general rule: the more constrained a computer vision context is, the more we can rely on those constraints to simplify the problem and the more reliable our final solution will be.

OpenCV is aimed at providing the basic tools needed to solve computer vision problems. In some cases, high-level functionalities in the library will be sufficient to solve the more complex problems in computer vision. Even when this is not the case, the basic components in the library are complete enough to enable creation of a complete solution of your own to almost any computer vision problem. In the latter case, there are several tried-and-true methods of using the library; all of them start with solving the problem using as many available library components as possible. Typically, after developing this first-draft solution, you can see where the solution has weaknesses and then fix those weaknesses using your own code and cleverness (better known as "solve the problem you actually have, not the one you imagine"). You can then use your draft solution as a benchmark to assess the improvements you have made. From that point, whatever weaknesses remain can be tackled by exploiting the context of the larger system in which your problem solution is embedded.

2.1.1Portability

OpenCV was designed to be portable. It was originally written to compile across Borland C++, MSVC++, and the Intel compilers. Th is meant that the C and C++ code had to be fairly standard in order to make cross-platform support easier. Figure 1-6 shows the platforms on which OpenCV is known to run. Support for 32-bit Intel architecture (IA32) on Windows is the most mature, followed by Linux on the same architecture. Mac OS X portability became a priority only aft er Apple started using Intel processors. (Th e OS X port isn't as mature as the Windows or Linux versions, but this is changing rapidly.) Th ese are followed by 64-bit support on extended memory (EM64T) and the 64-bit Intel architecture (IA64). Th e least mature portability is on Sun hardware and other operating systems. If an architecture or OS doesn't appear in OpenCV has been ported to almost every commercial system, from PowerPC Macs to robotic dogs. OpenCV runs well on

AMD’s line of processors, and even the further optimizations available in IPP will take advantage of multimedia extensions (MMX) in AMD processors that incorporate this technology.

2.1.2 Image Filtering:

It is a technique for modifying or enhancing an image. Image filtering is of two types. The one is linear image filtering, in which, the value of an output pixel is a linear combination of the values of the pixels of the input pixel's neighborhood. The second one is the non-linear image filtering, in which, the value of output is not a linear function of its input.

2.1.3 Image Transformation:

Image transformation generates "new" image from two or more sources which highlight particular features or properties of interest, better than the original input images. Basic image transformations apply simple arithmetic operations to the image data. Image subtraction is often used to identify changes that have occurred between images collected on different dates. Main image transformation methods are

- Hough Transform: used to find lines in an image
- Radon Transform: used to reconstruct images from fan-beam and parallel-beam projection data
- Discrete Cosine Transform: used in image and video compression
- Discrete Fourier Transform: used in filtering and frequency analysis
- Wavelet Transform: used to perform discrete wavelet analysis, denoise, and fuse images

2.1.4 Object Tracking:

Object tracking is the process of locating a object (or multiple objects) over a sequence of images. It is one of the most important components in a wide range of applications in computer vision, such as surveillance, human computer interaction, and medical imaging.

2.1.5 Feature Detection:

A feature is defined as an "interesting" part of an image and features are used as a starting point for many computer vision algorithms. Since features are used as the starting point and main

primitives for subsequent algorithms, the overall algorithm will often only be as good as its feature detector. Feature detection is a process of finding specific features of a visual stimulus, such as lines, edges or angle. It will be helpful for making local decisions about the local information contents (image structure) in the image.

2.2 OPENCV STRUCTURE AND CONTENT

OpenCV is broadly structured into five main components. The components contain basic image processing and higher-level computer vision algorithms; ML is the machine learning library, which includes many statistical classifiers and clustering tools. For storing and loading video and images, the HighGUI contains I/O routines and functions and CXCore contains the basic data structures and content.

2.2.1 CX CORE

This module contains basic data structures and basic functions used by other modules.

2.2.2 IMGPROC

Module contains image processing related functions such as linear, non-linear image filtering and geometrical image transformations etc.

At this point all of the basics at the systems disposal. The structure of the library as well as the basic data structures the system uses to represent images is well understood. HighGUI interface can actually run a program and display the results on the screen. Now that all this is understood ,primitive methods required to manipulate image structures are ready to learn some more sophisticated operations. The system will now move on to higher-level methods that treat the images as images, and not just as arrays of colored (or grayscale) values.

Smoothing, also called blurring, is a simple and frequently used image processing operation. There are many reasons for smoothing, but it is usually done to reduce noise or camera artifacts. Smoothing is also important when we wish to reduce the resolution of an image in a principled way (we will discuss this in more detail in the “Image Pyramids” section of this chapter).

2.2.3 VIDEO module contains motion estimation and object tracking algorithms. ML module contains machine-learning interfaces.

2.2.4 HighGUI

This module contains the basic I/O interfaces and multiplatform windowing capabilities.

The OpenCV functions that allow the user to interact with the operating system, the file system, and hardware such as cameras are collected into a library called HighGUI (which stands for “high-

level graphical user interface”). HighGUI allows the user to open windows, to display images, to read and write graphics-related fi les (both images and video), and to handle simple mouse, pointer, and keyboard events. It can also be used to create other useful doodads like sliders and then add them to the user’s windows. If one is a GUI guru in your window environment of choice, then you might find that much of what HighGUI offers is redundant. Yet even so you might find that the benefit of cross-platform portability is itself a tempting morsel.

However HighGUI library in OpenCV can be divided into three parts: the hardware part, the fi le system part, and the GUI part.

The hardware part is primarily concerned with the operation of cameras. In most operating systems, interaction with a camera is a tedious and painful task. HighGUI allows an easy way to query a camera and retrieve the latest image from the camera. It hides all of the nasty stuff , and that keeps us happy.

The file system part is concerned primarily with loading and saving images. One nice feature of the library is that it allows you to read video using the same methods you would use to read a camera. You can therefore abstract yourselves away from the particular device you are using and get on with writing interesting code. In a similar spirit, HighGUI provides you with a (relatively) universal pair of functions to load and save still images. These functions simply rely on the filename extension and automatically handle all of the decoding or encoding that is necessary.

The third part of HighGUI is the window system (or GUI). The library provides some simple functions that allows you to open a window and throw an image into that window. It also allows you to register and respond to mouse and keyboard events on that window. These features are most useful when trying to get off of the ground with a simple application. Tossing in some slider bars, which we can also use as switches, you find yourselves able to prototype a surprising variety of applications using only the HighGUI library.

However in the five main components CvAux was not included, but it is the one which contains both defunct areas (embedded HMM face recognition) and experimental algorithms (background/foreground segmentation). CvAux is not particularly well documented but it covers:

- Eigen objects, a computationally efficient recognition technique that is, in essence, a template matching procedure

- 1D and 2D hidden Markov models, a statistical recognition technique solved by dynamic programming

- Embedded HMMs (the observations of a parent HMM are themselves HMMs)

- Gesture recognition from stereo vision support

- Extensions to Delaunay triangulation, sequences, and so forth

- Stereo vision
- Shape matching with region contours
- Texture descriptors
- Eye and mouth tracking
- 3D tracking
- Finding skeletons (central lines) of objects in a scene
- Warping intermediate views between two camera views
- Background-foreground segmentation
- Video surveillance
- Camera calibration C++ classes (the C functions and engine are in CV) Some of these features may migrate to CV in the future; others probably never will.

2.3 Image and Video Transforms

2.3.1 Overview

In open cv these are methods for changing an image or video into an alternate representation of the data entirely. Perhaps the most common example of a transform would be a something like a Fourier transform, in which the image or video is converted to an alternate representation of the data in the original image or video. The result of this operation is still stored in an OpenCV "image" structure, but the individual "pixels" in this new image represent spectral components of the original input rather than the spatial components we are used to thinking about. There are a number of useful transforms that arise repeatedly in computer vision. OpenCV provides complete implementations of some of the more common ones as well as building blocks to help you implement your own image or video transforms.

2.3.2 Convolution

It is the basis of many of the transformations that are discussed in this chapter. What a particular convolution "does" is determined by the form of the Convolution kernel being used. Tis kernel is essentially just a fixed size array of numerical coefficients along with an anchor point in that array, which is typically located at the center. The size of the array is called the support of the kernel. The value of the convolution at a particular point is computed by first placing the kernel anchor on top of a pixel on the image with the rest of the kernel overlaying the corresponding local pixels

in the image. For each kernel point, you now have a value for the kernel at that point and a value for the image at the corresponding image point. Multiplication of these two together is then done and sum the result; this result is then placed in the resulting image at the location corresponding to the location of the anchor in the input image. This process is repeated for every point in the image by scanning the kernel over the entire image or video.

2.3.3 Scharr Filter

In openCV there are many ways to approximate a derivative in the case of a discrete grid. The downside of the approximation used for the Sobel operator is that it is less accurate for small kernels. For large kernels, where more points are used in the approximation, this problem is less significant. This inaccuracy does not show up directly for the X and Y filters used in cvSobel(), because they are exactly aligned with the x- and y-axes. The difficulty arises when you want to make image measurements that are approximations of directional derivatives (i.e., direction of the image gradient by using the arctangent of the y/x filter responses). To put this in context, a concrete example of where you may want image measurements of this kind would be in the process of collecting shape information from an object by assembling a histogram of gradient angles around the object. Such a histogram is the basis on which many common shape classifiers are trained and operated. In this case, inaccurate measures of gradient angle will decrease the recognition performance of the classifier. For a 3-by-3 Sobel filter, the inaccuracies are more apparent the further the gradient angle is from horizontal or vertical. OpenCV addresses this inaccuracy for small (but fast) 3-by-3 Sobel derivative filters by a somewhat obscure use of the special aperture_size value CV_SCHARR in the cvSobel() function. The Scharr fi lter is just as fast but more accurate than the Sobel fi lter, so it should always be used if you want to make image measurements using a 3-by-3 filter.

2.3.4 Laplace

The OpenCV Laplacian function implements a discrete analog of the Laplacian operator:

Because the Laplacian operator can be defined in terms of second derivatives, you might well suppose that the discrete implementation works something like the second-order Sobel derivative. Indeed it does, and in fact the OpenCV implementation of the Laplacian operator uses the Sobel operators directly in its computation.

The cvLaplace function takes the usual source and destination images as arguments as well as an aperture size. The source can be either an 8-bit (unsigned) image or a 32-bit (floating-point) image. The destination must be a 16-bit (signed) image or a 32-bit (floating-point) image. Th is aperture is precisely the same as the aperture appearing in the Sobel derivatives and, in effect, gives the size of the region over which the pixels are sampled in the computation of the second derivatives. The Laplace operator can be used in a variety of contexts. A common application is to detect "blobs." Recall that the form of the Laplacian operator is a sum of second derivatives along the x-axis and y-axis. This means that a single point or any small blob (smaller than the aperture) that is

surrounded by higher values will tend to maximize this function. Conversely, a point or small blob that is surrounded by lower values will tend to maximize the negative of this function. With this in mind, the Laplace operator can also be used as a kind of edge detector. To see how this is done, consider the first derivative of a function, which will (of course) be large wherever the function is changing rapidly. Equally important, it will grow rapidly as we approach an edge-like discontinuity and shrink rapidly as we move past the discontinuity. Hence the derivative will be at a local maximum somewhere within this range. Therefore we can look to the 0s of the second derivative for locations of such local maxima. Edges in the original image will be zeros of the Laplacian. Unfortunately, both substantial and less meaningful edges will be 0s of the Laplacian, but this is not a problem because we can simply f lter out those pixels that also have larger values of the first (Sobel) derivative

2.4 People counting technologies used before

In this section, a brief and point by point study of various approaches that can be adopted in people counting are presented.

2.4.1 Video Cameras

[1] describesan approach to people counting and localization using multiple video cameras. In this approach the focus lies on extracting the size and moving patterns of individuals passing. By means of motion histograms based on frame-differenced images, the histograms classify detected movements. Probabilistic correlation is applied to determine a people count. The results of multiple cameras are joined in order to form a movement vector for each individual recognized. In contrast, proposes a solution based on a single ceiling-mounted camera, which identifies people by background extraction of the camera image. A non-background "blob" is recognized, and its size is estimated and compared to previously established bounds of people's pixel dimensions. A people count is derived from the results of this analysis. The system reaches a claimed accuracy of 98.5%. The major disadvantage of a camera-based system is that it requires an ambient light source and relatively powerful computer resources to perform image processing

2.4.2 Ultrasonic Sensors

[2] introduce a system employing ultrasonic sensors. Per each observed area a three-node sensor cluster is established, whereby each sensor node mounts an ultrasonic sensor. Multiple clusters are joined to cover a wider area. Nodes in each cluster communicate sensor readings by an RF link to the cluster's coordinator node. The latter contributes its own sensor measurements. By means of a distributed algorithm, nodes decide on whether to count a detected person. The sensor nodes require clock synchronization at the millisecond level in order to correlate the data exchanged. Despite the availability of clock synchronization protocols this imposes a disadvantage to this

approach. The system achieves an overall counting accuracy of 90% using a probabilistic estimate of the total count, despite individual clusters achieving only around 50-70% accuracy.

2.4.3 Infrared Sensor

IR arrays combine a matrix of IR sensors to form array detectors. As the name suggests the sensor signals are provided as a matrix, where each element of the matrix corresponds to one IR sensor. Pattern recognition algorithms are able to detect people moving across the sensor's view at a claimed accuracy of 95%. According to [3], this holds true even if two pedestrian's paths cross, or people walk in parallel. IR arrays provide a cost-effective solution and also operate without any ambient light source. IR arrays are widely used in commercial systems.

2.4.4 Infrared Motion Sensors

In people counting system based on PIR motion detectors ,for each passage monitored, three PIR sensors are installed at a distance of 0.8m. The sensors are connected to a coordinator by a wireless RF link. Sensors detect motion events and send these data to the coordinator. The coordinator infers a people count from correlating the number, phase and time difference of peaks found in the signal. The system achieves a rate of 100% to detect the direction of movement, and accurately detects 89% of the number of people passing. PIR sensors provide an alternative to IR sensor arrays, however the cost and effort of employing multiple sensor nodes for each entry/exit point is a cost-side disadvantage. This system is based on just one PIR sensor and one sensor node per each observed entry/exit point. The system consists of camera, CO_2 and PIR sensors. It uses a Hidden Markovian Model (HMM) based on an Extended Kalman Filter (EKF) in order to derive building occupancy. The approach integrates historical data and current sensor readings to estimate the true state of the system, adjusting for sensor noise (false observations) and stochastic processes, for example uncertain people movement patterns.

The objective is to count the number of people who are at a bus stop using computer vision with OpenCV library.

2.5 Open CV Object Recognition

2.5.1Object Recognition with Machine Learning Algorithms

The first method for counting people in a video stream is to distinguish each individual object with the help of machine learning algorithms. For this purpose, the HOGdescriptor class has been implemented in OpenCV.

HOG (Histogram of Oriented Gradients) is a feature descriptor used in computer vision and image processing to detect objects. This technique is based on counting occurrences of gradient orientation in localized portions of an image.

HOGdescriptor implements a detector of histogram objects with oriented gradients. When utilizing HOG in object recognition, descriptors are classified based on supervised learning (support vector machines).

A Support Vector Machine, or SVM, is a supervised learning model that includes a set of associated learning algorithms. These algorithms are used for classification purposes. Coefficients calculated based on support vectors serve asthe basis for classification. A set of coefficients should be calculated on the basis of training data (an XML file). This file contains complete information on a model/classifier.

OpenCV includes two sets of pre-set nodes for people detection: Daimler People Detector and Default People Detector[6]

Here's an example of how HOG dsescriptor can be used (function interfaces can be found on the official OpenCV website):

```
cv::HOGDescriptor hog;
hog.setSVMDetector(cv::HOGDescriptor::getDefaultPeopleDetector());

// for every frame
std::vector<cv::Rect> detected;
hog.detectMultiScale(frame, detected, 0, cv::Size(8, 8), cv::Size(32, 32), 1.05f, 2);
```

According to [7] the effectiveness of recognition varies depending on the angle (top-view, side-view, profile-view). Side-view recognition is more unstable.

Fig 2.1: Top View Recognition [8]

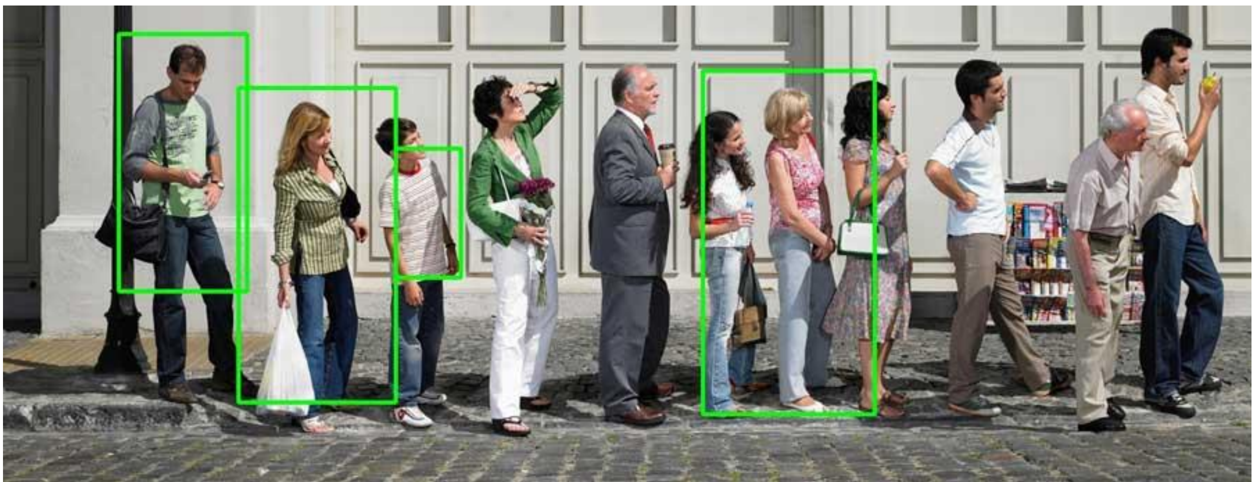

Fig.2.2 Side View Recognition [8]

According to [8] more accurate results can be obtained by modifying parameters of the detectMultiScale() function. Another advantage of this approach is that due to the learning ability, in some cases recognition can be improved to 90% or higher.

The next step is to recognize object movements. For more precise recognition, an object can be divided into parts (head, upper and lower torso, arms, and legs, for instance). In this way, the object will not be lost if it's overlapped by another object. Body parts can be identified with the help of the HOG descriptor, which contains a set of vertices for each body part. However, it has to be applied not to the whole image, but to a certain part with already identified object borders.

The disadvantage of this approach is that the algorithm slows down as the number of image vectors and their resolution increase. However, speed can be improved if all calculations are carried out using compute capacity of the video card.

2.5.2 Recognition of Moving Objects through Background Subtraction Algorithms

The second method for counting people in a video stream, which is more efficient, is based on recognition of moving objects. Considering the background to be static, two sequential frames are compared to identify differences. The simplest way is to compare two frames and generate a third one called a mask[9]

```
cv::absdiff(firstFrame, secondFrame, outputMaskFrame);
```

However, this method does not take into account such aspects as shades, reflections, position of light sources, or other changes to the environment, the mask might be quite inaccurate[12]. For such cases, the following three algorithms for automatic background detection and subtraction are implemented in OpenCV: BackgroundSubtractorMOG, BackgroundSubtractorMOG2, and BackgroundSubtractorGMG.

Here, the BackgroundSubtractorMOG2 method is used as an example. This method uses a Gaussian mixture model to detect and subtract the background from an image:

```
auto subtractor = cv::createBackgroundSubtractorMOG2(500, 2, false);

//for every frame

subtractor->apply(currentFrame, foregroundFrame);
```

For better recognition, any extra image noise that's present in a video has to be removed, especially in videos that have been recorded in low light. In some cases, a Gaussian smoothing can be used to slightly blur an image before subtracting its background[10]:

auto subtractor = cv::createBackgroundSubtractorMOG2(500, 2, false);

// for every frame

cv::GaussianBlur(currentFrame, currentFrame, cv::Size(10, 10), 0);

subtractor->apply(currentFrame, foregroundFrame);

cv::threshold(foregroundFrame, threshFrame, 10, 255.0f, CV_THRESH_BINARY);

After image noise and background have been subtracted, the image is displayed in black and white, where the background is black and all moving objects are white.

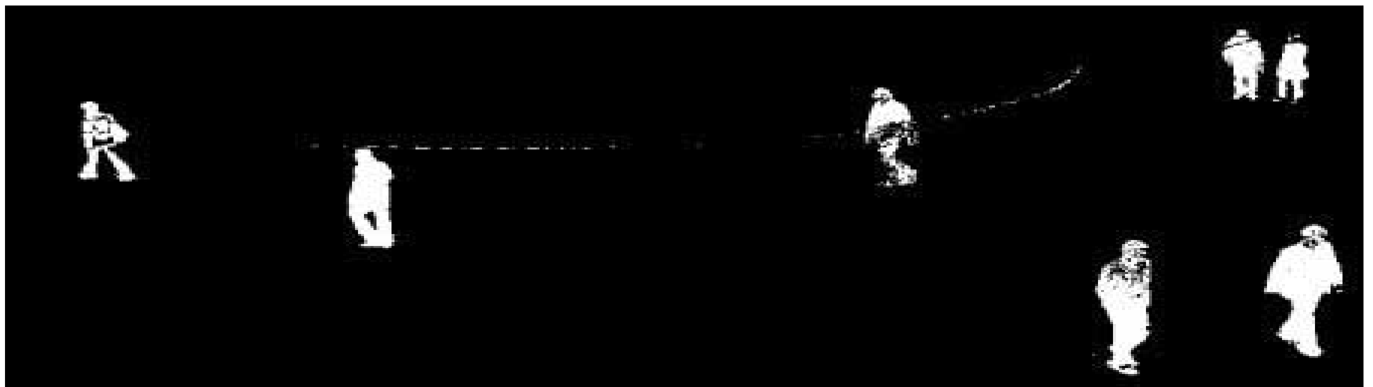

Fig. 3. BackgroundSubtractorMOG2 in use

In most cases, moving objects cannot be recognized completely and therefore have some empty space inside. To improve this, a triangle is created to cover the static space of an image. Then a morphological operation is performed to apply the triangle to the image so the empty space is filled:

```
// for every frame

cv::Mat structuringElement10x10 = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(10, 10));

cv::morphologyEx(threshFrame, maskFrame, cv::MORPH_OPEN, structuringElement10x10);

cv::morphologyEx(maskFrame, maskFrame, cv::MORPH_CLOSE, structuringElement10x10);
```

After that, the contours of all objects have to be identified and the boundaries of rectangles have to be saved to an array:

```
cv::TermCriteria termCrit(cv::TermCriteria::COUNT | cv::TermCriteria::EPS, 20, 0.03f);

// for every frame

cv::calcOpticalFlowPyrLK(prevFrame, nextFrame, pointsToTrack, trackPointsNextPosition, status, err, winSize, 3, termCrit, 0, 0.001f);
```

The Lucas-Kanade method estimates the optical flow for a sparse set of functions. OpenCV also provides the Farneback method, which allows searching for a solid optical flow. This method estimates the optical flow for all pixels in the frame.

In addition, OpenCV has the following five algorithms for object tracing with automatic detection of the contours of moving objects: MIL, BOOSTING, MEDIANFLOW, TLD, and KCF. All of them are implemented for the cv::Tracker class:

```
cv::Ptr<cv::Tracker> tracker = cv::Tracker::create("KCF");
tracker->init(frame, trackObjectRect);

// for every frame
tracker->update(frame, trackObjectRect);
```

As a result, each object is boxed in a rectangle. OpenCV traces whether this rectangle crosses an abstract line. In this way, the number of passers-by is counted.

Fig 2.3 System for Passers-by Counting in Operation [8]

To filter out small objects and to detect nearby moving objects, maximum and minimum width can be defined.

The method described above works best in good lighting and when there's considerable distance between the camera and an object. At the same time, recognition can be improved by defining object width at a given point on the plane, as sometimes a weakly recognized object can be perceived as multiple objects.

This algorithm can also be used for recognizing other moving objects such as cars.

References

[1] K. Terada, D. Yoshida, S. Oe, and J. Yamaguchi, A method of counting the passing people by using the stereo images, International conference on image processing, 0-7803-5467-2016

[2] Haritaoglu and M. Flickner, Detection and tracking of shopping groups in stores, Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition , 0-7695- 1272-0,2016.

[3] Gary Conrad and Richard Johnsonbaugh, A real-time people counter, Proceedings of the 1994 ACM symposium on Applied computing, 0-89791-647-6 ,1994[ROS94] : M. Rossi and A. Bozzoli, Tracking and Counting Moving People, IEEE Proc. of Int. Conf. Image Processing, ,2014

[4] [OpenCV Wiki] Open Source Computer Vision Library Wiki, http://opencvlibrary.sourceforge.net/.

[5] [OpenCV] Open Source Computer Vision Library (OpenCV), http://sourceforge.net/ projects/opencvlibrary/.

[6] D. Chen and G. Zhang, “A new sub-pixel detector for x-corners in cameracalibration targets,” WSCG Short Papers (2012).

[7] A. Colombari, A. Fusiello, and V. Murino, “Video objects segmentation by robust background modeling,” International Conference on Image Analysis and Processing, September 2007.

[8] Gary Bradski and Adrian Kaehler, O’REILLY Learning OpenCV Copyright © 2010 Printed in the United States of America.

[9] T. A. Clarke and J. G. Fryer, “The Development of Camera Calibration Methods and Models,” Photogrammetric Record 16 (1998).

[10] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski, “Selfsupervised monocular road detection in desert terrain,” Robotics: Science and Systems, Philadelphia, 2015

CHAPTER 3

3.1 Current methods of people counting

People counting using the proposed openCV involves counting people using raspberry pi camera algorithms and then send the results to a web server. A GPS module is also connected to the raspberry pi so that the system will be able to tell the exact location of the bus stop with certain number of people so as to make plan for the most effective route of the bus, saving so much time and costs. However currently the methods being used in people counting include video cameras, infrared sensors, infrared motion sensor and ultrasonic sensors.

3.2 Implementation of proposed system

3.2.1 Simulation using Fritzing simulation software

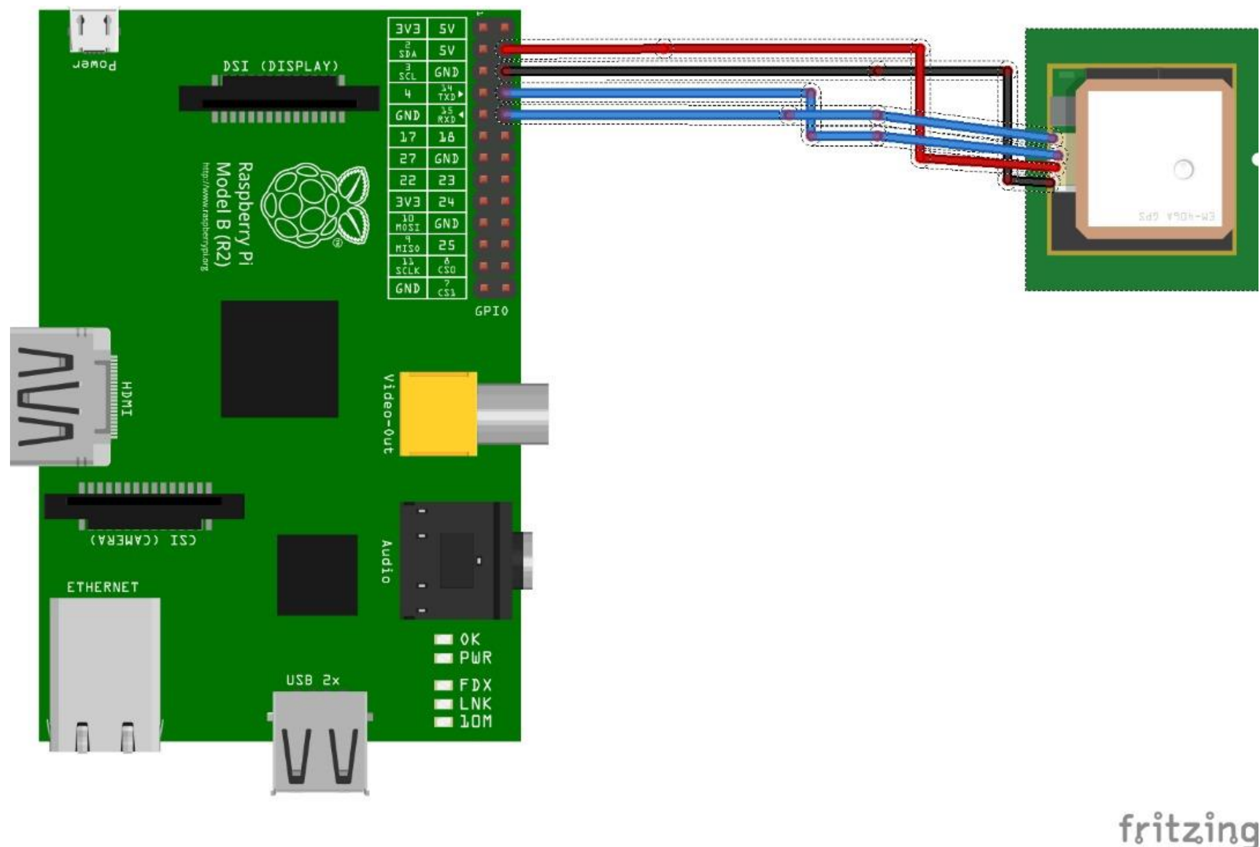

Fig 3.1 Simulation using Fritizing software

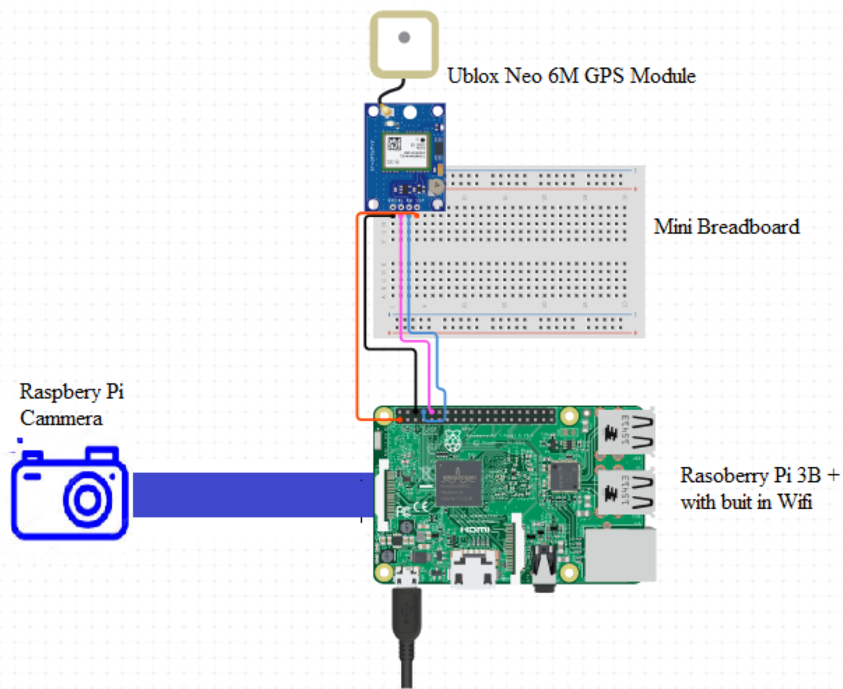

Fig 3.2 Proposed system circuit diagram

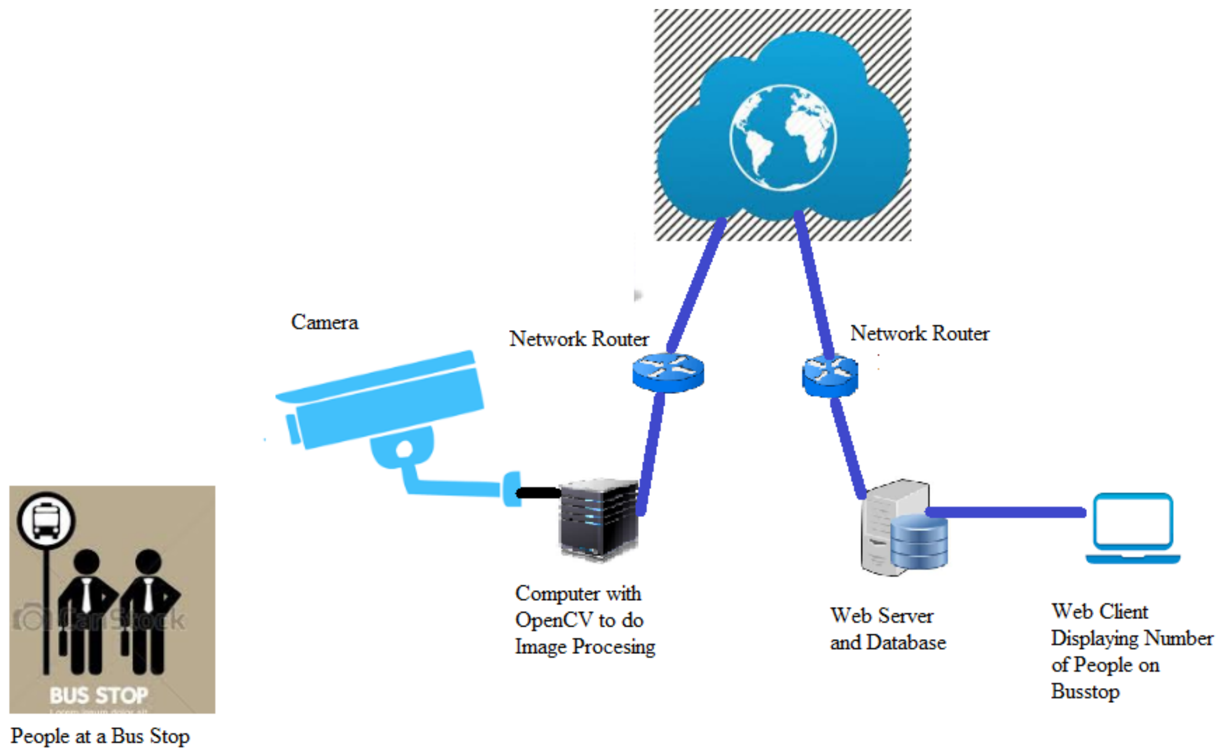

Fig 3.3 Design model

3.2 DESIGN AND IMPLEMENTATION

The prototype design was based on open source computer vision. Simulation of the circuit in Fig 3.2 was done using fritzing software as shown in fig 3.1. The whole idea was to come up with a system that is similar to that in figure 3.3. Simulation was done so as to ensure that all components are connected properly on the circuit board and also prevent short circuit which might cause damage to the components. The simulation software was also done for ease of troubleshooting. Using the Fritzing software helped in in reproducing the schematic simulated setup.

The real set up was based on the simulated setup. The raspberry pi was connected as the microcontroller. A web interface and database was created, and unlike in fig 3.3 where there is a real server, a laptop was used as both the database and the web server.

The camera was linked to raspberry pi for capturing the video, doing human detection and tracking. After capturing, detection and tracking, the video was then processed with different modules in open cv.

That information was used to plot graphs showing statistics for different bus stops.

What this project intended to do was to have a system that could, on its own calculate the the optimum routes, but because of the limitations of the the low processing power of the raspberry pi microcontroller that couldn't be achieved.

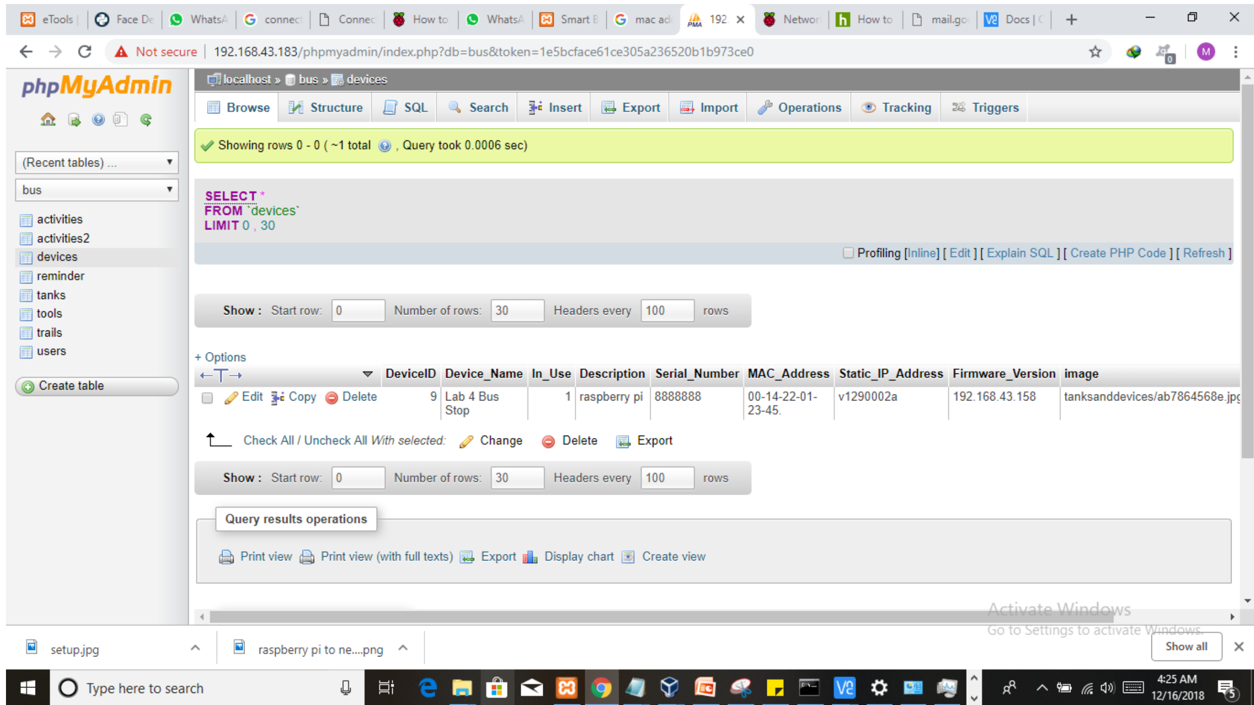

Fig 3.4 Gui interface first page for login in

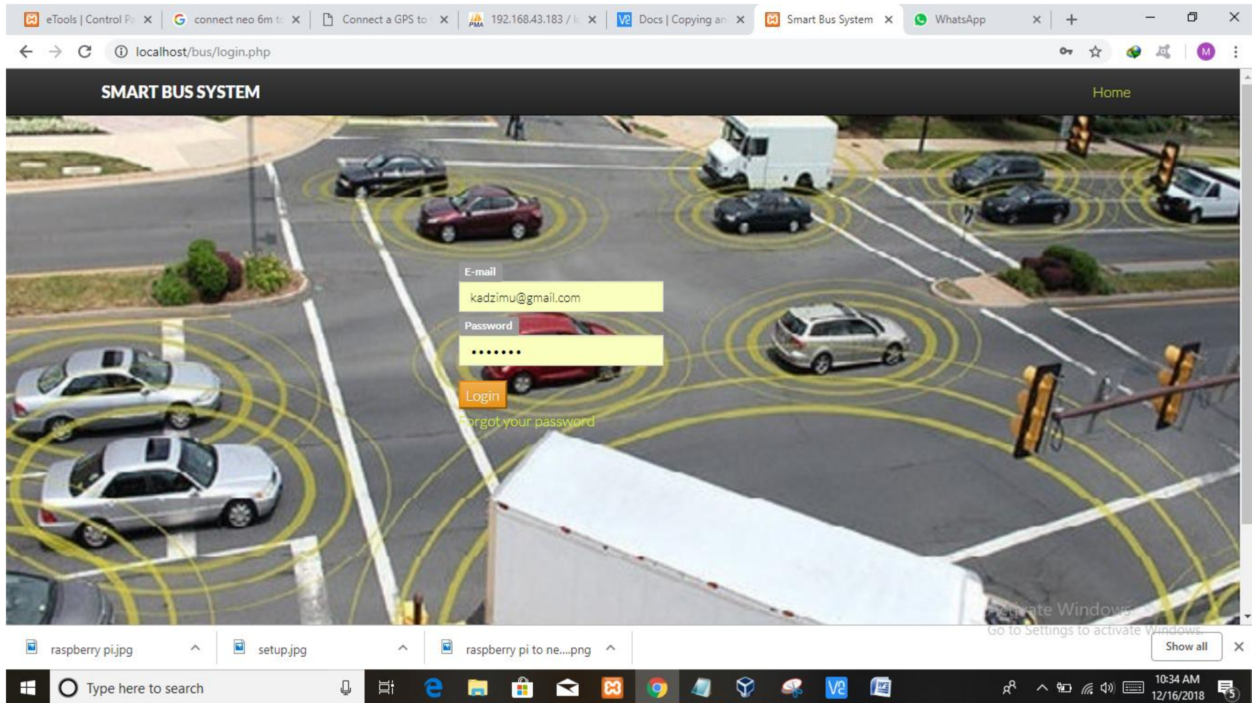

Fig 3.5 Login page input email and credentials

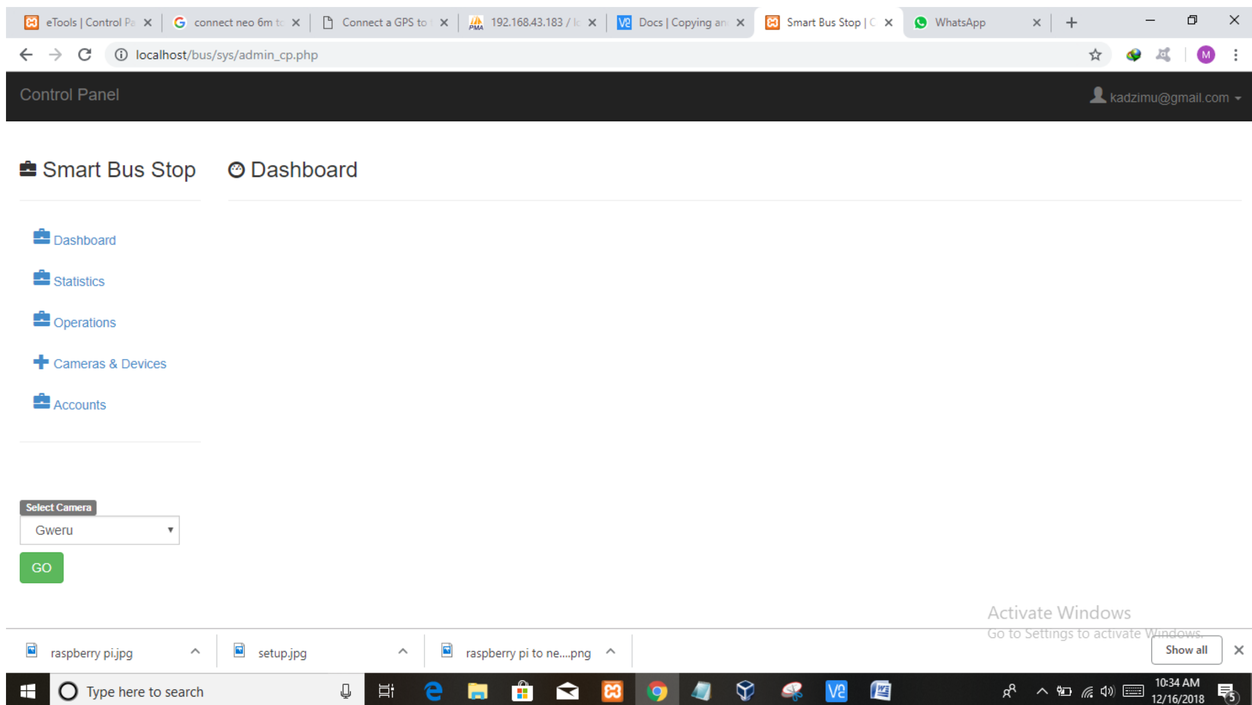

Fig 3.6GUI dash board

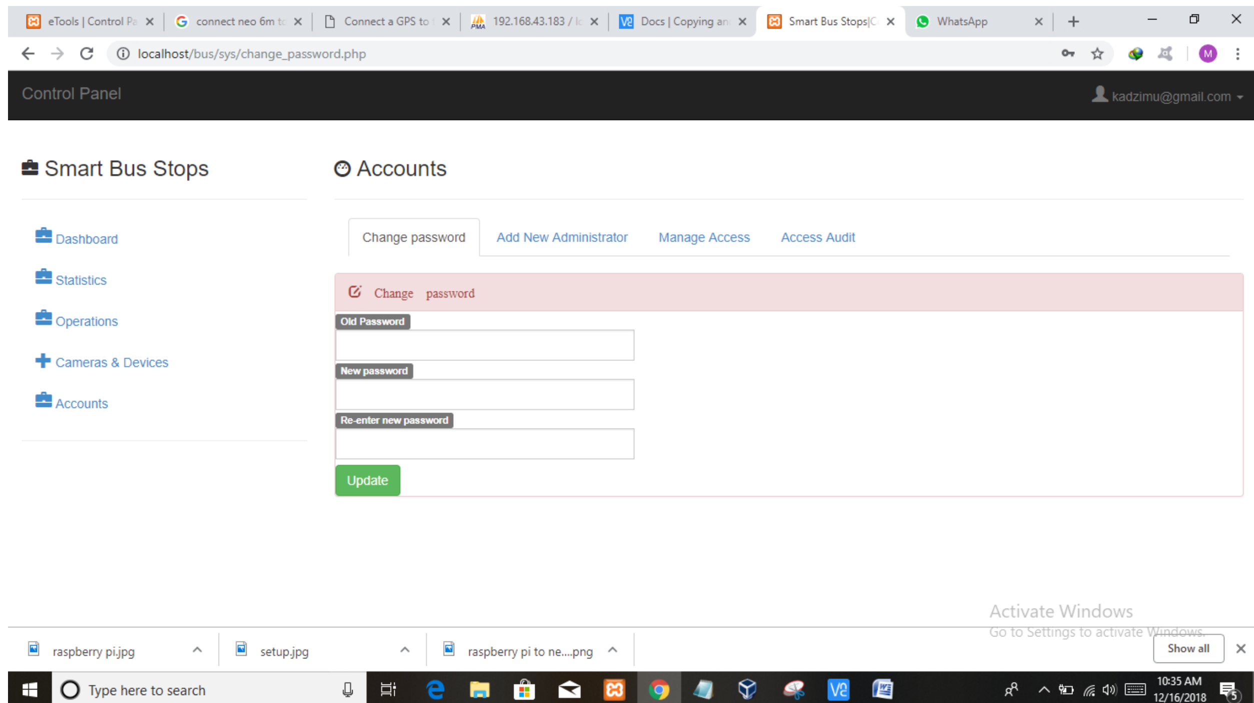

Fig 3.7 Managing accounts

Understanding object detection vs. object tracking

There is a fundamental difference between object detection and object tracking[1].

When object detection is applied this system will be determining where in an image/frame an object is. An object detector is also typically more computationally expensive, and therefore slower, than an object tracking algorithm [2]. Examples of object detection algorithms include Haar cascades, HOG + Linear SVM, and deep learning-based object detectors such as Faster R-CNNs, YOLO, and Single Shot Detectors (SSDs).

According to [3] an object tracker, on the other hand, will accept the input (x, y)-coordinates of where an object is in an image and will:

1. Assign a unique ID to that particular object

2. Track the object as it moves around a video stream, predicting the new object location in the next frame based on various attributes of the frame like gradient and optical flow.

Examples of object tracking algorithms include MedianFlow, MOSSE, GOTURN, kernalized correlation filters, and discriminative correlation filters, to name a few.

Combining both object detection and object tracking

[4] Open cv combine the concept of object detection and object tracking into a single algorithm, typically divided into two phases:

- **Phase 1 — Detecting:** During the detection phase the system will be running computationally more expensive object tracker to detect if new objects have entered the view, and also see if it can find objects that were "lost" during the tracking phase. For each detected object it creates or update an object tracker with the new bounding box coordinates.

- **Phase 2 — Tracking:** For each of the detected objects, an object tracker is created to track the object as it moves around the frame. The object tracker should be faster and more efficient than the object detector. It will continue tracking until it reached the N-th frame and then re-run the object detector. The entire process then repeats.

3.4 Deep Learning using YOLO

YOLOv3 is the latest variant of a popular object detection algorithm YOLO – You Only Look Once. The published model recognizes 80 different objects in images and videos, but most importantly it is super fast and nearly as accurate as Single Shot MultiBox (SSD).

Starting with OpenCV 3.4.2, you can easily use YOLOv3 models in your own OpenCV application.

3.4.1 How YOLO works

You can think of an object detector as a combination of a object locator and an object recognizer.

In traditional computer vision approaches, a sliding window was used to look for objects at different locations and scales. Because this was such an expensive operation, the aspect ratio of the object was usually assumed to be fixed.

Early Deep Learning based object detection algorithms like the R-CNN and Fast R-CNN used a method called Selective Search to narrow down the number of bounding boxes that the algorithm had to test.

Another approach called Overfeat involved scanning the image at multiple scales using sliding windows-like mechanisms done convolutionally.

This was followed by Faster R-CNN that used a Region Proposal Network (RPN) for identifying bounding boxes that needed to be tested like in fig 3.9. By clever design the features extracted for recognizing objects, were also used by the RPN for proposing potential bounding boxes thus saving a lot of computation.

YOLO on the other hand approaches the object detection problem in a completely different way. It forwards the whole image only once through the network. SSD is another object detection algorithm that forwards the image once though a deep learning network, but YOLOv3 is much faster than SSD while achieving very comparable accuracy. YOLOv3 gives faster than realtime results on a M40, TitanX or 1080 Ti GPUs. [6]

3.5 YOLO detection the objects in a given image.

First, it divides the image into a 13×13 grid of cells. The size of these 169 cells vary depending on the size of the input. For a 416×416 input size that we used in our experiments, the cell size was 32×32. Each cell is then responsible for predicting a number of boxes in the image[7,8]

For each bounding box, the network also predicts the confidence that the bounding box actually encloses an object, and the probability of the enclosed object being a particular class.

Most of these bounding boxes are eliminated because their confidence is low or because they are enclosing the same object as another bounding box with very high confidence score. This technique is called **non-maximum suppression**.

The authors of YOLOv3, Joseph Redmon and Ali Farhadi, have made YOLOv3 faster and more accurate than their previous work YOLOv2. YOLOv3 handles multiple scales better. They have also improved the network by making it bigger and taking it towards residual networks by adding shortcut connections.

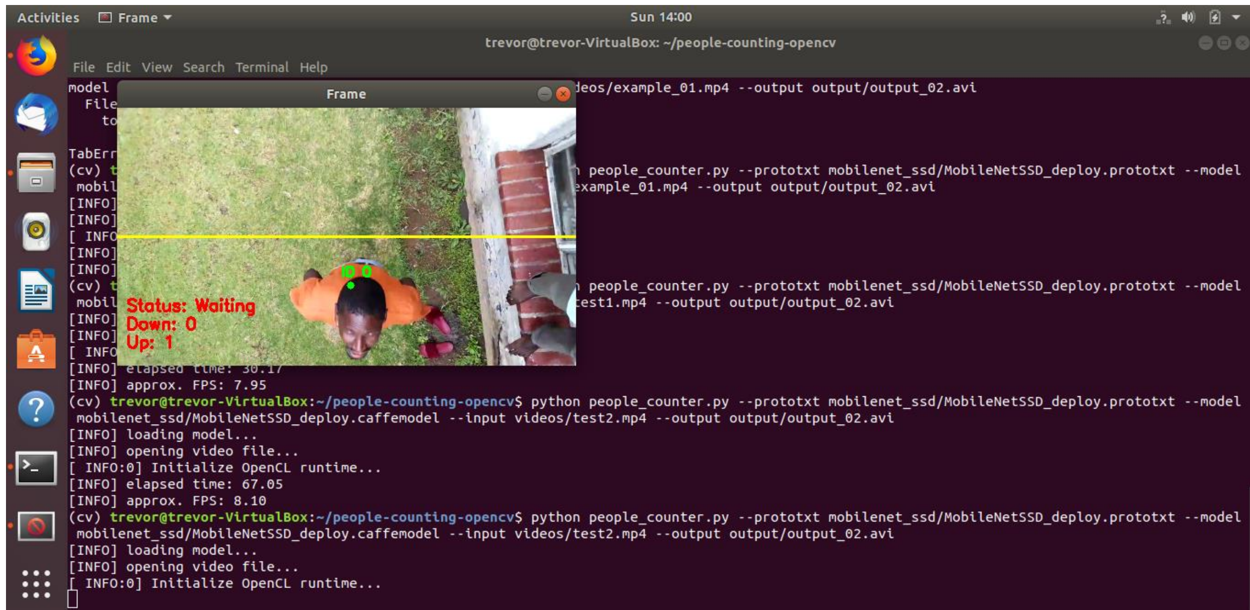

Fig 3.9 YOLO detection

3.7 Why use OpenCV for YOLO ?

Here are a few reasons why YOLO was used for open cv

1. **Easy integration with an OpenCV application**: If your application already uses OpenCV and you simply want to use YOLOv3, you don’t have to worry about compiling and building the extra Darknet code.
2. **OpenCV CPU version is 9x faster**: OpenCV’s CPU implementation of the DNN module is astonishingly fast. For example, Darknet when used with OpenMP takes about 2 seconds on a CPU for inference on a single image. In contrast, OpenCV’s implementation runs in a mere 0.22 seconds! Check out table below.
3. **Python support**: Darknet is written in C, and it does not officially support Python. In contrast, OpenCV does. There are python ports available for Darknet though.

References

[1] [OpenCV] Open Source Computer Vision Library (OpenCV), http://sourceforge.net/projects/opencvlibrary/.

[2] D. Chen and G. Zhang, “A new sub-pixel detector for x-corners in cameracalibration targets,” WSCG Short Papers (2012).

[3] A. Colombari, A. Fusiello, and V. Murino, “Video objects segmentation by robust background modeling,” International Conference on Image Analysis and Processing, September 2007.

[4] Gary Bradski and Adrian Kaehler, O’REILLY Learning OpenCV Copyright © 2010 Printed in the United States of America.

[5] T. A. Clarke and J. G. Fryer, “The Development of Camera Calibration Methods and Models,” Photogrammetric Record 16 (1998).

[6] K. Terada, D. Yoshida, S. Oe, and J. Yamaguchi, A method of counting the passing people by using the stereo images, International conference on image processing, 0-7803-5467-2016

[7] Haritaoglu and M. Flickner, Detection and tracking of shopping groups in stores, Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition , 0-7695- 1272-0,2016.

[8] Gary Conrad and Richard Johnsonbaugh, A real-time people counter, Proceedings of the 1994 ACM symposium on Applied computing, 0-89791-647-6 ,1994[ROS94] : M. Rossi and A. Bozzoli, Tracking and Counting Moving People, IEEE Proc. of Int. Conf. Image Processing, ,2014

[9] [OpenCV Wiki] Open Source Computer Vision Library Wiki, http://opencvlibrary.sourceforge.net/.

CHAPTER 4

4. Results and Analysis

4.1 1Introduction

This chapter seeks to provide the testing results f the prototype and also to evaluate if the project managed to address all its objectives of providing a real world solution for the bus infrastructure system[1]

4.2 Functional Test Results

These are results that reflect of show the compliance or level of accuracy of the systemon meeting the required objectives

4.2.1 People Counting

The results observed from the prototype represented the actual expectations .The system managed to detect and count the people entering and leaving the bus stop demacated area.Both the functions of detecting and that of tracking were correctly executed and the results are clearly shown in the pictures below.

The first picture is a live feed picture of people entering a demarcated vicinity representing a bus stop.

The system successfully allocated the detected person an ID (in green) which is the used to count entry and exit.

Fig 4.1 People counting using open cv

4.2.2 Database Updating

The system accurately managed to update the people count numbers to the server .This is is he second testing point of the system which is also critical as the real time updates are the ones that ae used as the statistical base for the overall bus sytemmanagement.

The picture below show the people count at a specific busstop id linked to the coordinates in the database .

GPS coordinates were successfully sent hence the identification of the busstop from which the data is coming from.

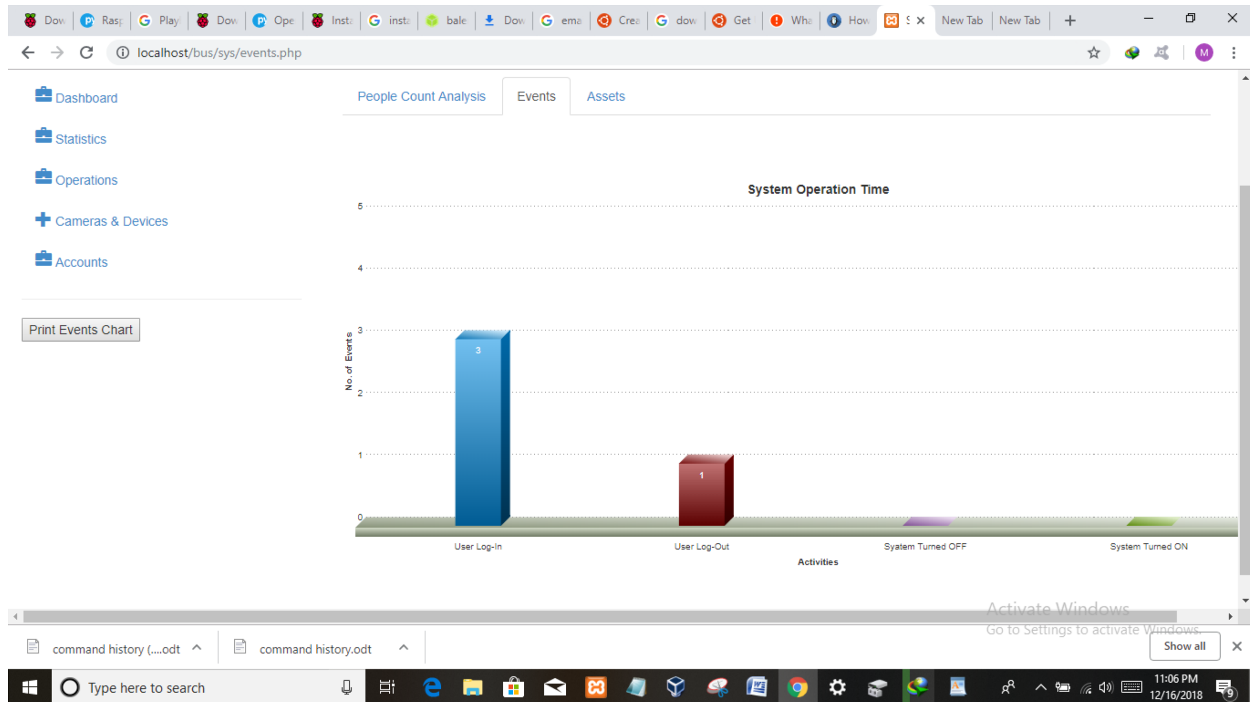

Fig 4.2 Statistical Results from the people counter

4.2.3GUI Design and function

The GUI graphical user interface was successful designed and also linked to the raspberry pi system hence all people counting results were successfully sent to the server forfurther computations of Fig 4.3 should show the total count of the people onsite.

The gui managed to store the user login records which is vital for security purposes

The graphs below also shows the device inventory for remote monitoring

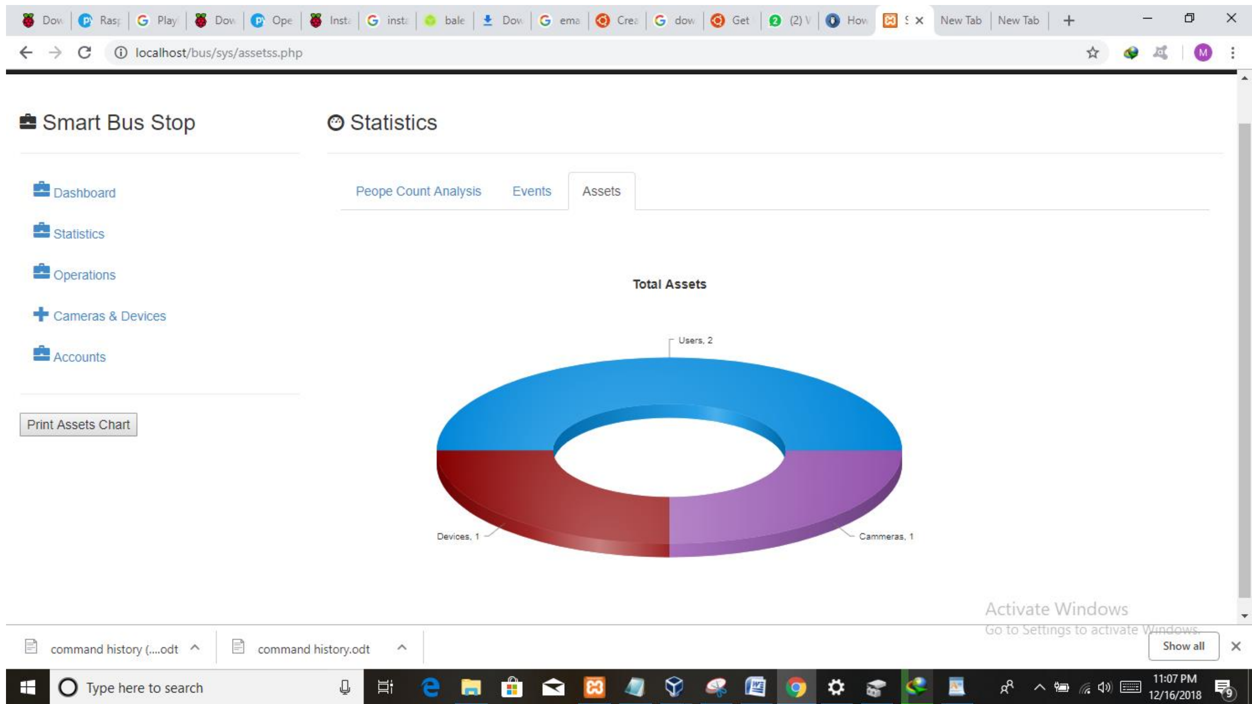

Fig 4.3 Total count onsite

References.

[1] Gary Bradski and Adrian Kaehler, O'REILLY Learning OpenCV Copyright © 2010 Printed in the United States of America

CHAPTER 5

5 CONCLUSION

5.1 Introduction

In this chapter a review of the research project is conducted ogether with the highlighting of the system improvements that can be done .a summarized conclusion is also give in this chapter

5.2 Real World Operation analysis

A prototype or model is just and abstract of the intended real system hence this prototype was just a limited simple model [1].The system was designed using simple microcontroller (raspberry pi instead of the real word processing units with lots of processing power to conduct efficient deep learning,detection computations

A simplepi camera was used instead of a high resolution and night vision enabled camera inr to cut costs .Also the GPS used was not efficiently compatible with the pi camera .The Raspberry Pi microcontroller was too weak to process image recognition algorithms reading to its eventual malfunctioning .

The read/write rates were very high for the class 10 SD card hence but the alternative laptop simulation what to be finally done after realizing .

However the obtained results were good enough to justify that this is a sustainable project which can be implemented everywhere since It doesn't have many components meaning less weak points .The Google map API attachment and route plot were not achieved because they needed also more processing power for them to be live .

For it be justifiably operational in real world the system should needs a camera and processor combination that gives good frame rate for detection and tracking .

The use of a cloud based processing system is now favorable which simply replaces the small microcontroller and can be used with IP camera which can utilize the existing mobile network infrastructure.

5.3 Recommendation

More powerful processing machines must be used instead of to Raspberry pi or personal laptops to enable the use of more detailed logarithms . The Raspberry Pi microcontroller was too weak to process image recognition algorithms reading to its eventual malfunctioning ..also the gps used was not efficiently compatible with the pi camera hence more advance systems can be employed to achieve the best result . .

The system can be greatly improved by the addition of the initially proposed googlemaps and automatic bus routing output.

The system can also be enhanced by the inclusion of mobile app for the users to check if their routes are being prioritized especially in high density areas. Passengers can know if there is a bus dispatched for them.

5.4 Summary and conclusion

This system is verygood for busoperators tooptimie their services and also good for the people who get the service .Due to this fact we can generally the project was a success in terms of objective attainment however some critical challenges that were due to lack of powerful microcontrollers on the market resulted in the failure of the major prototype to run for a long time before crushing .This does not affect the feasibility since there are a lot of powerful machines and shared resources that can process all the information efficiently

References

[1] Gary Bradski and Adrian Kaehler, O'REILLY Learning OpenCV Copyright © 2010 Printed in the United States of America.

APPENDICES

Appendix A:Source Code

```
# USAGE

# To read and write back out to video:

# python people_counter.py --prototxt mobilenet_ssd/MobileNetSSD_deploy.prototxt \

#                    --model    mobilenet_ssd/MobileNetSSD_deploy.caffemodel    --input
videos/example_01.mp4 \

#                    --output output/output_01.avi

#

# To read from webcam and write back out to disk:

# python people_counter.py --prototxt mobilenet_ssd/MobileNetSSD_deploy.prototxt \

#                    --model mobilenet_ssd/MobileNetSSD_deploy.caffemodel \

#                    --output output/webcam_output.avi


# import the necessary packages

from pyimagesearch.centroidtracker import CentroidTracker

from pyimagesearch.trackableobject import TrackableObject

from imutils.video import VideoStream

from imutils.video import FPS

import numpy as np

import argparse

import imutils

import time

import dlib

import cv2

import datetime

import requests
```

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--picamera", type=int, default=-1,
help="whether or not the Raspberry Pi camera should be used")
ap.add_argument("-p", "--prototxt", required=True,
help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
help="path to Caffe pre-trained model")
ap.add_argument("-i", "--input", type=str,
help="path to optional input video file")
ap.add_argument("-o", "--output", type=str,
help="path to optional output video file")
ap.add_argument("-c", "--confidence", type=float, default=0.4,
help="minimum probability to filter weak detections")
ap.add_argument("-s", "--skip-frames", type=int, default=30,
help="# of skip frames between detections")

args = vars(ap.parse_args())




# initialize the list of class labels MobileNet SSD was trained to
# detect
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
"bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
"dog", "horse", "motorbike", "person", "pottedplant", "sheep",
"sofa", "train", "tvmonitor"]
```

```
# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# if a video path was not supplied, grab a reference to the webcam
if not args.get("input", False):
print("[INFO] starting video stream...")
#vs = VideoStream(src=0).start()
#time.sleep(2.0)
vs = VideoStream(usePiCamera=args["picamera"] > 0).start()
time.sleep(2.0)

# otherwise, grab a reference to the video file
else:
print("[INFO] opening video file...")
vs = cv2.VideoCapture(args["input"])

# initialize the video writer (we'll instantiate later if need be)
writer = None

# initialize the frame dimensions (we'll set them as soon as we read
# the first frame from the video)
W = None
H = None

# instantiate our centroid tracker, then initialize a list to store
```

```
# each of our dlib correlation trackers, followed by a dictionary to
# map each unique object ID to a TrackableObject
ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
trackers = []
trackableObjects = { }

# initialize the total number of frames processed thus far, along
# with the total number of objects that have moved either up or down
totalFrames = 0
totalDown = 0
totalUp = 0
Longtitude = "29.839095"
Latitude = "-19.515886"
Device_Name = "Lab 4 Bus Stop"
Status_Level = 1
Down_Value = 0
Up_Value = 0
RemoteIP = "192.168.43.183"

# start the frames per second throughput estimator
fps = FPS().start()

# loop over frames from the video stream
while True:
# grab the next frame and handle if we are reading from either
# VideoCapture or VideoStream
frame = vs.read()
```

```
frame = frame[1] if args.get("input", False) else frame

# if we are viewing a video and we did not grab a frame then we
# have reached the end of the video
if args["input"] is not None and frame is None:
		break

# resize the frame to have a maximum width of 500 pixels (the
# less data we have, the faster we can process it), then convert
# the frame from BGR to RGB for dlib
frame = imutils.resize(frame, width=400)
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

# if the frame dimensions are empty, set them
if W is None or H is None:
		(H, W) = frame.shape[:2]

# if we are supposed to be writing a video to disk, initialize
# the writer
if args["output"] is not None and writer is None:
		fourcc = cv2.VideoWriter_fourcc(*"MJPG")
		writer = cv2.VideoWriter(args["output"], fourcc, 30,
		   (W, H), True)

# initialize the current status along with our list of bounding
# box rectangles returned by either (1) our object detector or
# (2) the correlation trackers
```

```
status = "Waiting"
rects = []

# check to see if we should run a more computationally expensive
# object detection method to aid our tracker
if totalFrames % args["skip_frames"] == 0:
                        # set the status and initialize our new set of object trackers
                        status = "Detecting"
                        trackers = []

                        # convert the frame to a blob and pass the blob through the
                        # network and obtain the detections
                        blob = cv2.dnn.blobFromImage(frame, 0.007843, (W, H), 127.5)
                        net.setInput(blob)
                        detections = net.forward()

                        # loop over the detections
                        for i in np.arange(0, detections.shape[2]):
                           # extract the confidence (i.e., probability) associated
                           # with the prediction
                           confidence = detections[0, 0, i, 2]

                           # filter out weak detections by requiring a minimum
                           # confidence
                           if confidence > args["confidence"]:
                                    # extract the index of the class label from the
                                    # detections list
```

```
			idx = int(detections[0, 0, i, 1])

			# if the class label is not a person, ignore it
			if CLASSES[idx] != "person":
				continue

			# compute the (x, y)-coordinates of the bounding box
			# for the object
			box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
			(startX, startY, endX, endY) = box.astype("int")

			# construct a dlib rectangle object from the bounding
			# box coordinates and then start the dlib correlation
			# tracker
			tracker = dlib.correlation_tracker()
			rect = dlib.rectangle(startX, startY, endX, endY)
			tracker.start_track(rgb, rect)

			# add the tracker to our list of trackers so we can
			# utilize it during skip frames
			trackers.append(tracker)

# otherwise, we should utilize our object *trackers* rather than
# object *detectors* to obtain a higher frame processing throughput
else:
		# loop over the trackers
		for tracker in trackers:
```

```
                        # set the status of our system to be 'tracking' rather
                        # than 'waiting' or 'detecting'
                        status = "Tracking"

                        # update the tracker and grab the updated position
                        tracker.update(rgb)
                        pos = tracker.get_position()

                        # unpack the position object
                        startX = int(pos.left())
                        startY = int(pos.top())
                        endX = int(pos.right())
                        endY = int(pos.bottom())

                        # add the bounding box coordinates to the rectangles list
                        rects.append((startX, startY, endX, endY))

        # draw a horizontal line in the center of the frame -- once an
        # object crosses this line we will determine whether they were
        # moving 'up' or 'down'
        cv2.line(frame, (0, H // 2), (W, H // 2), (0, 255, 255), 2)

        # use the centroid tracker to associate the (1) old object
        # centroids with (2) the newly computed object centroids
        objects = ct.update(rects)

        # loop over the tracked objects
```

```
for (objectID, centroid) in objects.items():
			# check to see if a trackable object exists for the current
			# object ID
			to = trackableObjects.get(objectID, None)

			# if there is no existing trackable object, create one
			if to is None:
				to = TrackableObject(objectID, centroid)

			# otherwise, there is a trackable object so we can utilize it
			# to determine direction
			else:
				# the difference between the y-coordinate of the *current*
				# centroid and the mean of *previous* centroids will tell
				# us in which direction the object is moving (negative for
				# 'up' and positive for 'down')
				y = [c[1] for c in to.centroids]
				direction = centroid[1] - np.mean(y)
				to.centroids.append(centroid)

				# check to see if the object has been counted or not
				if not to.counted:
					# if the direction is negative (indicating the object
					# is moving up) AND the centroid is above the center
					# line, count the object
					if direction < 0 and centroid[1] < H // 2:
						totalUp += 1
```

```
                                        to.counted = True
                                        Up_Value = totalUp
                                        payload = {'value1': Longtitude, 'value2': Latitude, 'value3':
Device_Name, 'value4': Up_Value,'value5':Down_Value, 'value6':Status_Level}
                              requests.get('http://%s/bus/charts/arduino_data.php'   %   RemoteIP,
params=payload)


                                      # if the direction is positive (indicating the object
                                  # is moving down) AND the centroid is below the
                                  # center line, count the object
                                  elif direction > 0 and centroid[1] > H // 2:
                                        totalDown += 1
                              to.counted = True
                                        Down_Value = totalDown
                                        payload = {'value1': Longtitude, 'value2': Latitude, 'value3':
Device_Name, 'value4': Up_Value, 'value5':Down_Value, 'value6':Status_Level}
                                        requests.get('http://%s/bus/charts/arduino_data.php'     %
RemoteIP, params=payload)



                         # store the trackable object in our dictionary
                         trackableObjects[objectID] = to


                         # draw both the ID of the object and the centroid of the
                         # object on the output frame
                         text = "ID {}".format(objectID)
                         cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
                         cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)
```

```
# construct a tuple of information we will be displaying on the
# frame

info = [
        ("Up", totalUp),
        ("Down", totalDown),
        ("Status", status),
]

# loop over the info tuples and draw them on our frame
for (i, (k, v)) in enumerate(info):
        text = "{}: {}".format(k, v)
        cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
           cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

# check to see if we should write the frame to disk
if writer is not None:
        writer.write(frame)

# show the output frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop
if key == ord("q"):
        break
```

```
# increment the total number of frames processed thus far and
# then update the FPS counter
totalFrames += 1
fps.update()

# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# check to see if we need to release the video writer pointer
if writer is not None:
writer.release()

# if we are not using a video file, stop the camera video stream
if not args.get("input", False):
vs.stop()

# otherwise, release the video file pointer
else:
vs.release()

# close any open windows
cv2.destroyAllWindows()
```