TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Abu Ishtiaque Md Fahim 195458

Camera Based Person Tracking and Location Detection on Embedded Hardware

Master's thesis

Supervisor: Mairo Leier
PhD

Co-supervisor: Uljana Reinsalu
PhD

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Abu Ishtiaque Md Fahim 195458

Kaamerapõhine inimeste jälgimine ja asukoha tuvastamine sardriistvaral

magistritöö

Juhendaja: Mairo Leier
Doktorikraad
Kaasjuhendaja Uljana Reinsalu
Doktorikraad

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Abu Ishtiaque Md Fahim

10.05.2021

Abstract

This thesis is focused on real-time human location detection and tracking in an indoor area using camera and embedded hardware. The target system is developed using convolutional neural network that facilitate person detection methods. The project is developed mainly in three steps – detecting human, determining the location and track the location along with time frame. The total area is determined using the angle and manually measured height of the camera and image depth by comparing it with the frame size from the camera feed. OpenCV library is used to detect humans and objects. Then image pixels containing the detected human is being compared against the area analysed in the first step in a continuous fashion for each frame to draw the final path.

This thesis is written in English and is 55 pages long, including 7 chapters, 26 figures and 2 table.

List of abbreviations and terms

ANN	Artificial Neural Network
SVM	Support Vector Machine
HCI	Human-computer Interaction
I/O	Input and Output
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
DSP	Digital Signal Processing
LBS	Location Based Service
GPS	Global Positioning System
IDE	Integrated Development Environment
CNN	Convolutional Neural Network
DNN	Deep Neural Network
OPENCV	Open-Source Computer Vision Library
SSD	Single-Shot Detector
R-CNN	Regions with Convolutional Neural Networks
HOG	Histogram of Oriented Gradients
ML	Machine Learning
AI	Artificial Intelligence
CSV	Comma-separated Values
FPS	Frames Per Seconds

Table of contents

Author's declaration of originality 3
Abstract 4
List of abbreviations and terms 5
Table of contents 6
List of figures 8
List of tables 9
1 Introduction 10
1.1 Background 12
1.2 Motivation 13
1.3 Objective 13
2 Hardware, Software and Other Tools 14
2.1 Embedded Hardware 15
2.1.1 Typical Embedded System Architecture 15
2.1.2 Comparison of Different Embedded Hardware 16
2.1.3 Advantages and Disadvantages 16
2.2 Camera for Detection and Tracking in Indoor 17
2.3 Object Detection in Embedded Hardware 18
2.4 Software and Libraries 18
2.5 Methodologies for Person Detection 21
3 Existing Methods and Challenges 19
3.1 Some Methods Overview for Object Detection 21
3.1.1 Fast R-CNN 24
3.1.2 Faster R-CNN 24
3.1.3 SSD 24
3.1.4 MobileNets 24
3.2 Existing Works on Person Detection and Tracking 25
3.3 Challenges of Using Embedded Hardware 19
3.3.1 Stability 20
3.3.2 Safety 20

3.3.3 Security 20
3.3.4 Launch Phase 20
3.3.5 Design Limitations 21
3.4 Drawbacks of Existing Works 29
4 Detection and Tracking 29
4.1 Person Detection 30
4.2 Determining Location 31
4.2.1 Defining Location 32
4.2.2 Approach of Location Calculation 33
4.2.3 Implementation of Location Calculation 38
4.3 Tracking Person Based on Detected Location 40
4.3.1 Finding Individuals in Consecutive Frames 40
4.3.2 Tracking Mechanism 41
4.3.3 Log for Tracking Data 43
5 Experiments and Results 45
5.1 Limitation and Improvements 49
5.1.1 Data Size 49
5.1.2 Partial Body Detection 50
5.1.3 Detection and Flicking 50
5.1.4 Cloud Storage Option and Security 51
5.1.5 Hardware Limitations 51
5.1.6 Works Only on Plain Ground 51
6 Summary 52
7 References 53
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis 55

List of figures

Figure 1. Generating Small Segments [1] 11
Figure 2. Feature Extraction [1] 11
Figure 3. Single Bounding Rectangle [1] 12
Figure 4. Basic Architecture of Embedded System [4] 15
Figure 5. Flow Chart of Human Detection [8] 23
Figure 6. Person detection with opencv and mobilenet 31
Figure 7. 2D plain of the indoor location where the camera is placed. 33
Figure 8. Horizontal angle representation of the room in 2D plain where camera is in centre. 34
Figure 9. Vertical angle representation of the room assuming the Z axis on the 2D plain. 35
Figure 10. Y or distance calculation from the camera 37
Figure 11. X Distance Calculation 38
Figure 12. Angle Calculation 39
Figure 13. Distance Calculation Between Person and Camera 39
Figure 14. Calculate Distance 40
Figure 15. Person, Location, Track Classes 41
Figure 16. Location Update and Tracking 42
Figure 17. Draw Track Line 43
Figure 18. Person status check and Log Writing 43
Figure 19. Sample csv file 44
Figure 20. CSV data for each row. 44
Figure 21. Sample row represents the data of a person. 44
Figure 22. Data Save in CSV file 45
Figure 23. Detected person shown by a green rectangle on the frame. 46
Figure 24. Calculated location is shown of the detected person in white text. 47
Figure 25. Real time location tracking by a red line 48
Figure 26. Detected persons tagged with unique ids. 49

List of tables

Table 1. Hardware Comparison [7] 16
Table 2. Faster R-CNN and SSD Accuracy Comparison [26] 25

1 Introduction

During the most recent years, there has been a quick and successful deployment on computer vision research. Many portions of this achievement have come from accepting and adapting the methods of machine learning. Other achievements come from the development of new models for specific computer vision related issues. The field of object detection has gained a huge progress. Real-time object or human detection seems emerging a significant trend around the world among the researchers, data scientists, mass industries from smart cities to retail for the purpose of surveillance. In computer vision, object detection is a technique in which a system can detect, trace, and locate the object from a video or an image feed. One of the extraordinary attributes about the object detection is that it can recognize the class of the objects and identify the specific coordinates of the location from the given image. Classes of the objects can be person, car, chair etc. By drawing a squared bounding box around the object, the location of the object is indicated. It is not certain that the bounding box may point out the exact position of the objects. The performance of the detection algorithm depends on the efficiency of the detected object from a picture or video. For example, Face Detection is an example of object detection. These detection algorithms can be trained from the scratch or it might be pre-trained. In many cases, pre-trained weights from a pre-trained model are used then it is being fine-tuned according to the purpose of the requirements or different use cases.

Generally, object detection is done in three steps:

- Generates the small segments in the input as shown in the figure 1, the large set of bounding boxes are spanning the full image. [1]

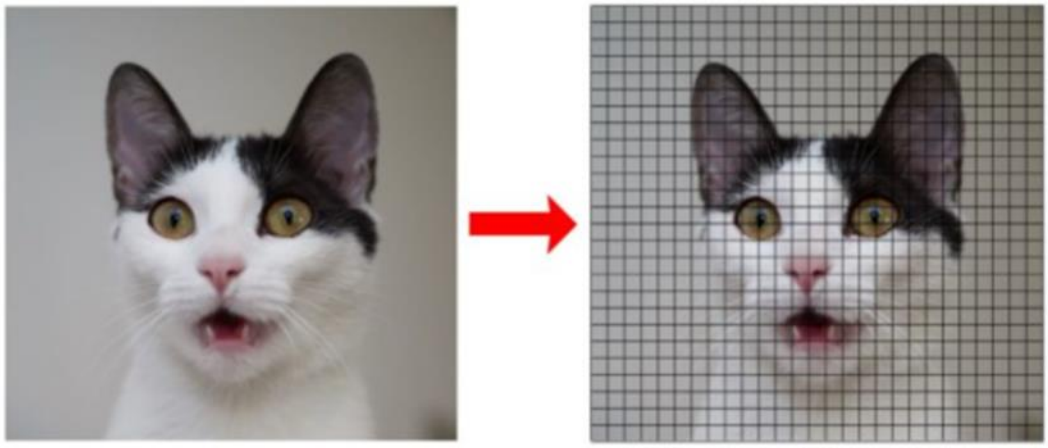

Figure 1. Generating Small Segments [1]

- Feature extraction is performed for each segmented rectangular area to predict if the rectangle contains a valid object as shown in figure 2. [1]

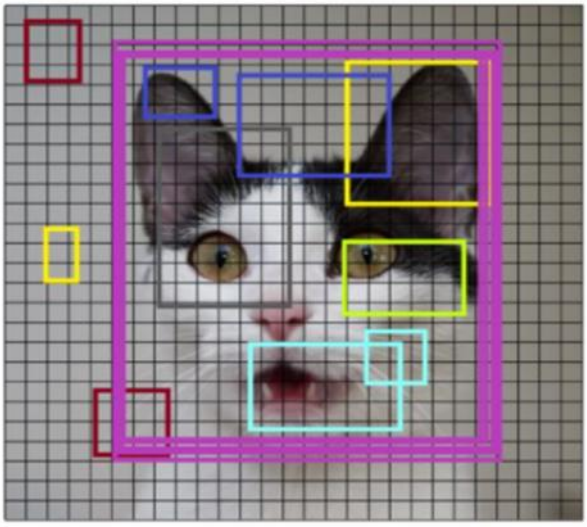

Figure 2. Feature Extraction [1]

- Overlapping boxes are joined into a single bounding rectangle as shown in figure 3 (Non-Maximum Suppression) [1]

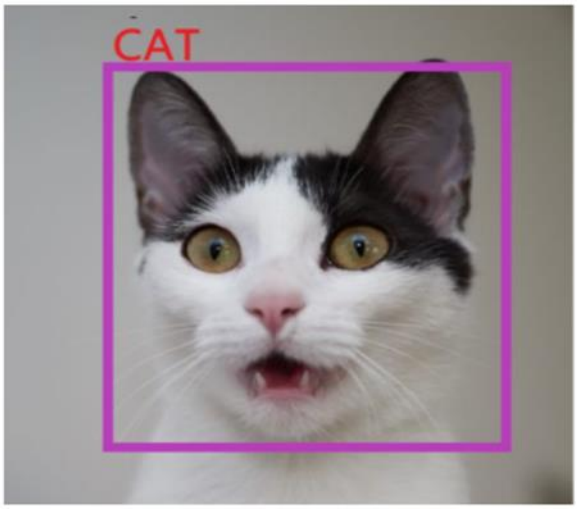

Figure 3. Single Bounding Rectangle [1]

1.1 Background

In computer vision systems, object detection is the primary task which is performed to take as many information as possible regarding the object which is being detected and the scene where the object is placed [2]. Once the object is detected, it is possible to get more information such as recognizing the specific instance (e.g. identifying the face), track the object in a sequence of the image (e.g. tracking the face in a video feed), determining the further information about the object (e.g. colour of the subject) while it is feasible to (a) presume the location of the object in a scene and (b) to get some idea about further information related to the scene (e.g., scene type, indoor or outdoor) [2].

Object detection has several used cases. Some popular applications are: (i) HCI (Human Computer Interface) (ii) consumer electronics (iii) transportation, (iv) robotics (v) security. These applications have different types of requirements such as processing time, detection, occlusions resistance whether pose changes [2]. Although many of these applications take in to account the detection of a single object class from one view [2]. On the other hand, some includes the detection of multiple classes for example human, cat, vehicles from many views. Most programs, in general, can detect only a single object type from a limited range of views and poses [2].

Several research papers and surveys on detection and recognition have been published during previous years. There are four main issues that are related to object detection.

Object localization, which entails determining the position and size of a single object instance that is known to exist in the image.

Object presence classification, which correlate with determining at least one object of a class which is in an image.

Object recognition, which determines a specific object that is present in the image.

View and pose estimation, which determines the view and pose of the object.

1.2 Motivation

The motivation of this project is to implement a camera-based person location detection and tracking system which will be used in an indoor environment. The application of this system may use in an hospital where mentally disabled patients can roam freely inside the area. To detect the location and track the patients is the goal here. In the indoor environment, Wrist bands with Global Positioning System (GPS) cannot work properly to track each and individual person. By using this system, we can track each patient centrally. With the help of cameras, no wearable devices will be required to track and detect any patient location. The system will keep patient's movement log file as a record so that hospital authority can take further necessary steps for the patients. With the help of Machine Learning (ML) and Artificial Intelligence (AI) and python, many projects can be done in embedded hardware whereas previously it required advanced computational capable machine which are very costly. For this project, our target is to implement a method which can detect the location from the camera after that our goal is to track the detected person from the camera feed.

1.3 Objective

In this thesis, the aim is to implement a video surveillance method using an embedded system for an indoor environment to detect human location and track movements. The main challenge is to use an efficient yet simple location detection and tracking method so that the hardware can take the load. The Open-Source Computer Vision (OpenCV) library can be used along with Python to achieve that goal. The OpenCV function draws a rectangle around the moving object and give us an indication of what is moving in front

of the camera. The current objective is to detect human movement, but the scope of detecting other objects also exist. On the other hand, the hardware is relatively weak for which no intensive operation can be done, making it challenging to get a real time movement detection. Another goal of this implementation is to detect the coordinate of the moving object to get a trail. The main challenge of this is to determine the distance of two point on the visual frame by mapping the real-life dynamic distance and the available pixels.

2 Hardware, Software and Other Tools

Embedded systems are used commonly nowadays. To enable embedded systems to comprehend and acknowledge the visual information is one of the ways to accomplish this. As a result, academia and industry have put a lot of work into developing algorithms that give embedded systems and machines in general the ability to perceive visual information [3]. Many of these algorithms are aimed at allowing embedded systems to identify specific objects using vision. Object detection and tracking in embedded systems can be accomplished by video and image processing. It is used in a variety of embedded applications, including computer vision and image processing, bioinformatics, surveillance, and artificial intelligence. Object detection is the method of analysing an image to see if it has a certain object and where in the image it is located.

Today many embedded systems and products exist that are capable of object detection ranging from digital cameras to autonomous robots [3]. To perform detection, a variety of algorithms are used, which has some advantages and drawbacks. The most widely used detection and classification algorithms are focused on ANN (Artificial Neural Networks), SVM (Support Vector Machines) and the newly introduced Viola-Jones detection framework [23], which was created initially for face detection, but has recently gained popularity in several other applications as well. These algorithms have proven versatile as they have been called upon to solve very different classification problems with successful results.

2.1 Embedded Hardware

Embedded system has two parts. (i) Embedded Hardware and (ii) Embedded Software. Embedded Hardware are based on microcontroller and microprocessor. Memory bus, Controller, Input and Output (I/O) are also included in embedded hardware. Embedded software includes embedded operating systems, different drivers, and applications. Basically, there are two types of architecture that are used in embedded systems. These are Harvard architecture and Von Neumann architecture. Architecture includes Sensor, Memory processor, DAC (Digital to Analog Converter), ADC (Analog to Digital Converter), and actuators etc.

2.1.1 Typical Embedded System Architecture

The figure 4 demonstrates the overview of typical architecture of embedded systems:

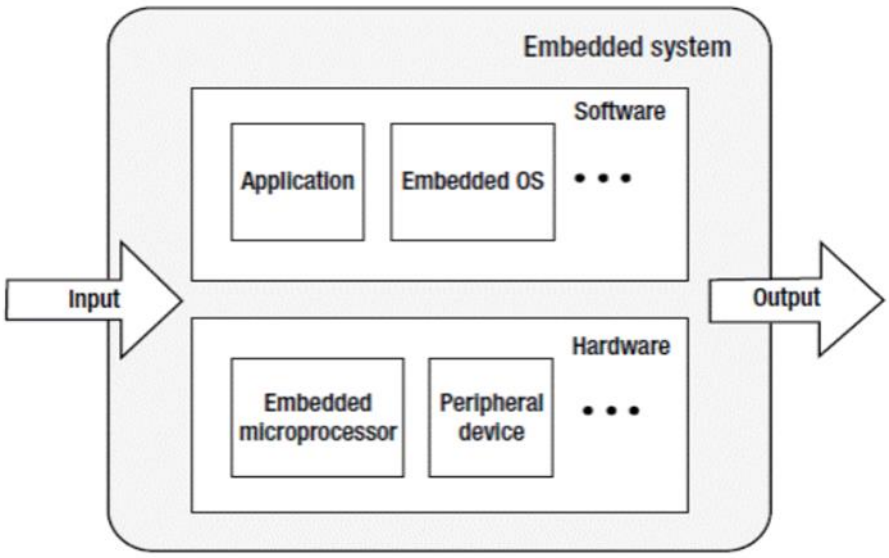

Figure 4. Basic Architecture of Embedded System [4]

Input and output are features of every open system, and the embedded system is no different. In an embedded system, the hardware and software often work together to deal with a variety of external input signals and to output the processing results in some manner. The output of a sensor circuit in another embedded system could be used as the input signal. The output could be a record or file for a database, or it could be sound, light, electricity, or some analogue signal.

2.1.2 Comparison of Different Embedded Hardware

Some commonly used embedded hardware are Nvidia jetson nano, Google coral, Khadas Vim3 Edge TPU. These are some specialized hardware which works as accelerators for neural networks. Here, these hardware are compared based on the performance of some pre-trained neural network such as MobileNet, MobileNet-SSD (Single Shot Detector), EffiicientNet, ResNet. These neural networks are mainly used is object detection related tasks. Table 1 demonstrates some performance comparison of two devices.

Table 1. Hardware Comparison [7]

Model	**Framework**	**Nvidia Jetson Nano**	**Google Coral**
EfficientNet-B0 (224x224)	TensorFlow	216 FPS	200FPS
ResNet-50 (244x244)	TensorFlow	36 FPS	18 FPS
MobileNet-v2 (300x300)	TensorFlow	64 FPS	130 FPS
SSD Mobilenet-V2 (300x300)	TensorFlow	39 FPS	48 FPS
Inception V4 (299x299)	PyTorch	11 FPS	9 FPS

Nvidia Jetson nano is performing better than google coral from the above comparison except for MobileNet-v2. Moreover, Jetson nano supports most models because all frameworks such as TensorFlow, Caffe, PyTorch, YOLO, MXNet, and others use the CUDA GPU support library at a given time. Whereas the Google Coral works with special pre-compiled TensorFlow Lite networks [7]. If the topology of the neural network and its required operations can be described in TensorFlow it may work well on Google Coral. However, with its sparse 1 Gbyte RAM, memory shortage can still be an issue [7]. Jetson nano also has immense support available compared with google coral. For Khadas VIM 3, it has 8- and 16-bit neural network processing unit, performance is expected to be faster than Google coral.

2.1.3 Advantages and Disadvantages

Because of the minimal size, cost per item is low which makes it mass-production friendly. It has a high reliability and stability because they are designed for a fixed set of

tasks. As embedded systems are designed to perform a specific array of tasks, it ensures the best possible performance optimization. Due to the size and nature, embedded systems are fast and consumes marginally less power. Embedded systems are designed in a way where it requires the optimization of the usage of the available resources and the product quality is enhanced because of it. Last but not the least, embedded systems are small which makes them highly portable.

Embedded Systems needs to be pre-configured and are not dynamic in nature which means there are no scope of improvement/upgrade once the system is configured. Keeping back-ups of embedded files are also a challenge. Due to being tiny and sensitive, embedded systems are harder to maintain. Because of the same reason, troubleshooting is difficult for an embedded system. Transferring data from one embedded system to another is troublesome because both the systems need to be of the similar compatibility and capability. As each embedded system are made for particular tasks, the hardware usability is limited to those tasks and can't be used for anything else.

2.2 Camera for Detection and Tracking in Indoor

Uses of location-based services is increasing in industry and research. For location-based services, GPS is commonly used. Nowadays, GPS is available in smartphone, smartwatch, fitness bands and in many other gadgets. Unfortunately, GPS is not suitable to locate and track people in indoor condition with a decent accuracy. GPS signals can be lost due to attenuation effects of roof tops or walls leads a very low accuracy due to multipath blockage. Moreover, indoor location-based service requires higher accuracy and precision guarantees than outside services. Error should not exceed a few meters. Otherwise, services could provide a result where the location of the person may quite far away from the actual position. There are many existing indoor positioning techniques which can be categorized by their expenses and precision. Available indoor positioning systems which are solely purposed for this such as ultra-sonic system has several components only purposed for determining the multiple targets position in indoor environments. These have a precision rate but in terms of cost, they are quite expensive than cameras. Expensive infrastructure is needed. With the help of simple RGB cameras, these expensive gadgets can be replaced for indoor positioning and tracking. With the help of embedded hardware and camera, person location can be detected in an indoor

location and the scope of tracking is also available. From multiple views with multiple angles, people location can be detected inside the room and it is easily trackable.

2.3 Object Detection in Embedded Hardware

Object detection can be useful for a wide range of embedded systems ranging from cell phones to surveillance cameras, robotics, and autonomous vehicles However, object detection in embedded systems is a challenging task primarily for two reasons: real-time performance constraints and low power. Real-time constraints are set by the operational environment of the embedded system and set a lower limit of the performance of the object detection system. Low power concerns many other aspects of processing and computing but is of utmost importance to embedded systems because often they must operate on battery power for many hours. As such, dedicated hardware architectures that target embedded object detection must follow some basic design principles to facilitate real-time constraints and low power.

In many cases the targeted embedded systems may facilitate other applications other than object detection as well. One such case is the cell phone where face detection just one of many applications that run on the embedded system. Therefore, hardware architectures must keep a low area overhead by using the minimum amount of hardware resources for the specific functionality. Furthermore, the same hardware design may not be suitable for all embedded systems as each has its own set of specifications and platform implementation. As such, hardware architectures for embedded object detection must have a modular and a regular design to maintain scalability and should provide a degree of run time reconfigurability to handle different embedded applications. Finally embedded hardware architectures for object detection must provide efficient memory management, with parallel memory access and predictable access patterns.

2.4 Software and Libraries

For developing software-based solutions, there are two approaches. One is implementing a new method. Other one is reusing the existing code by implementing and improving it according to the project purpose. Implementing a project from a scratch is time consuming and requires more tuning than using other codes. Using existing code may not

be directly suitable for the project. To get the best result, one way is to use an open-source software. License of the software allows to download existing source code and allow customization in the according to the vision of the project. For implementing this project, Python programming language and PyCharm IDE is used. OpenCV library is used in this project. Some other libraries of python such as Math, Imutils, NumPy is also used, and some definition of this libraries is described below.

Python is a high-level interpreted language. This language is used in web development, GUI development. This language also can perform complex machine learning related tasks and enable to build a prototype as quickly as possible to experiment the product for machine learning purposes [5]. For the development of this project, python version 3.7 is used.

PyCharm is a cross-platform IDE which is specifically designed for python language. This provides good code analysis, have graphical debugger, version control system and also supports web development with Django and Anaconda for data science.

Anaconda is a distribution for python language. Anaconda is suitable for the purpose of machine learning application, data science, prediction analysis.

OpenCV is a library which available in cross-platform. OpenCV is mainly used in computer vision related applications, image processing, image or video capture and analysis. This library also has features like object detection, face detection. OpenCV 4.4.0 version is used in this project.

Other libraries of python such as math, imutils, numpy, csv, datetime, time, pip are also used during the implementation of the project.

2.5 Challenges of Using Embedded Hardware

Embedded software is always a constituent of a larger system, for instance, a digital watch, a smartphone, a vehicle, or automated industrial equipment [10]. Such embedded systems must respond in real time under all conditions within the timeframe defined by the designer, and they must work with limited memory, computing capacity, and energy supply. Furthermore, embedded software must be resistant to improvements in its

operating system, which may include processors, sensors, and hardware modules. Portability and autonomy are two other difficult criteria for embedded applications.

2.5.1 Stability

The value of stability cannot be overstated. Unexpected behaviour from an embedded device is unacceptable and dangerous. End users expect embedded systems to behave consistently under all conditions and to last for an extended period without requiring maintenance.

2.5.2 Safety

Safety is a special feature of embedded systems due to their primary application associated with lifesaving functionality in critical environments [10]. In terms of consistency, research, and technical experience, the Software Development Life Cycle (SDLC) for embedded software is marked by more stringent specifications and limitations.

2.5.3 Security

In the modern world, security has been a hot topic. The associated risks increase exponentially, specifically as IoT devices gain popularity across the world and become more interconnected. Since modern home appliances such as induction cookers, refrigerators, and washing machines come equipped with networking features by nature, the Internet of Things is now vulnerable to hacking attacks.

2.5.4 Launch Phase

In embedded system development, time-to-market and time-to-revenue have always been challenging indicators, particularly in the IoT sector. As a result, the applications and services that will support the zillions of IoT devices anticipated by 2020 are only in the design stage. Due to the rapid growth of the IoT industry, fabrication of hardware components housing embedded devices necessitates intense incorporation and flexibility. Furthermore, with the longer lifetime of IoT devices, component designers must consider potential upgrades and launches.

2.5.5 Design Limitations

For decades, the problems in embedded system design have been based on the same restricting requirements:

• Small size.

• Consumes low energy.

• Long-term, reliable production without the need for repairs.

Designers are now under pressure from the competition to squeeze both computing capacity and battery life into smaller rooms, which is often a trade-off. Finally, the market for highly scalable processor families ranging from low-cost and ultra-low-power to high-performance and highly configurable processors with forward-compatible instruction sets is increasing, depending on IoT applications.

3 Existing Methods and Challenges

Researcher has been working on object detection and tracking for many years. This sector is improving in a fast pace every day. Several object detection and tracking methods has been introduced by using ML, ANN, Image processing and so on. With the help from these algorithms object detection can be done in much ease. Some algorithms are required high end devices to run and some of them are specifically designed for embedded hardware. Most of the object detection algorithms and methods usually detects the object and track the object without defining the exact location of object inside the frame. By location, it means what is the distance of the object from the camera. In this chapter, we explained some of the Convolutional Neural Networks (CNN) overview for object detection and some works of previous researchers regarding this field.

3.1 Methodologies for Person Detection

Person detection is the task of locating all instances of human beings present in an image, and it has been most widely accomplished by searching all locations in the image, at all

possible scales, and comparing a small area at each location with known templates or patterns of people [6]. A human is generally detected by segmenting motion in a video image. Most conventional approaches for human detection are background subtraction, optical flow, and spatio-temporal filtering method. They are outlined in the following subsections.

Background subtraction is a common technique for detecting a foreground target in a surveillance camera scene by segmenting it. The camera could be stationary, translational only, or handheld. Background subtraction is a pixel-by-pixel or block-by-block method of detecting moving particles from the difference between the present frame and the reference frame. The reference frame is often referred to as a 'background picture,' a 'background model,' or a 'environment model.' A strong background model must be able to adjust to the changing conditions of complex scenes. This may be accomplished by updating the history details at regular intervals.

Optical flow is a vector-based method for estimating video motion by comparing points on objects through image frames (s). Optical flow is used to explain the consistent motion of points or features between image frames under the assumption of brightness constancy and spatial smoothness [8]. Optical flow-based motion segmentation detects moving regions in an image series by analysing the flow vectors of moving objects over time. One of the biggest advantages of using optical flow is that it can handle many cameras and object movements at the same time, making it suitable for crowd monitoring and environments with a lot of movement. Except in the presence of camera motion, optical flow-based approaches may be used to track independently moving objects. Aside from image noise, colour, and non-uniform illumination, most flow computing approaches have high computational specifications and are vulnerable to motion discontinuities. Because of the algorithm's complexities and the need for precise measurements, a real-time implementation of optical flow would often necessitate specialized hardware.

The movement or gesture is defined by the entire 3D Spatio-temporal data volume spanned by the moving person in the picture series for motion recognition dependent on Spatio-temporal analysis. To define motion's spatio-temporal distributions, these approaches usually treat it as a whole. The filter exhibits high responses at motion regions due to the derivative activity on the temporal axis. These responses were then used to create thresholds, resulting in a binary motion mask, which was then grouped into spatial

histogram bins. This function compactly encodes motion and its related spatial detail, which is useful for surveillance videos in the far-field and medium-field. These methods are quick and simple to execute since they are built on simple convolution operations. They are especially useful in low-resolution or low-quality video situations where extracting other features like optical flow or silhouettes is difficult. Methods focused on Spatio-temporal motion are better at capturing both spatial and temporal information about gait motion. Their benefit is that they are easy to execute and have low computational complexity. They are, however, vulnerable to noise and differences in action timing. Figure 5 demonstrates the flow chart of human detection from a video sequence.

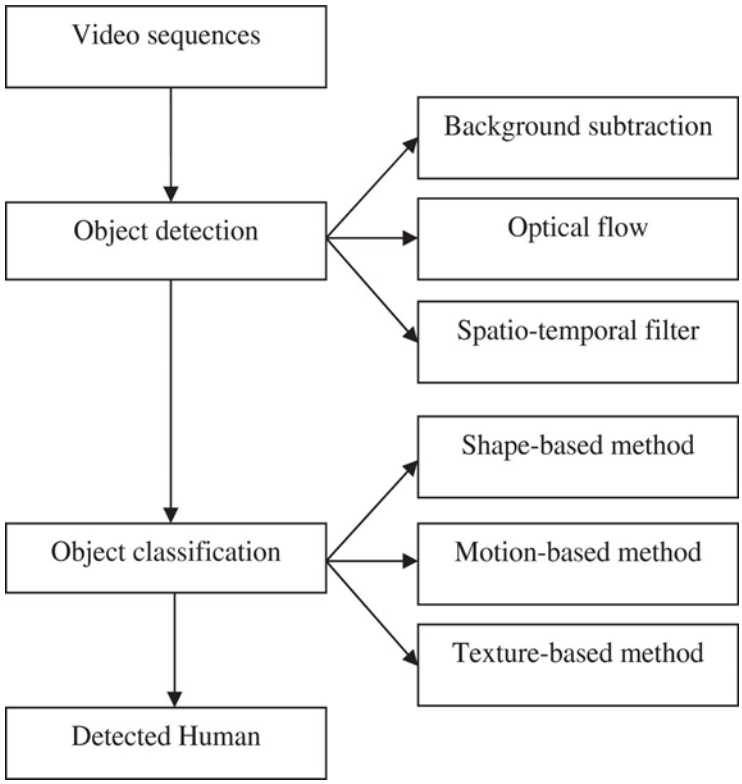

Figure 5. Flow Chart of Human Detection [8]

3.2 Overview of Some Neural Networks for Object Detection

There are many methods and algorithms for object detection. Here, we are providing some neural network overview which is used to object detection purposes.

3.2.1 Fast R-CNN

Fast Region-Based Convolutional Network System, or Fast R-CNN, is a training algorithm for object detection written in Python and C++ (Caffe). This algorithm primarily addresses the shortcomings of R-CNN and SPPnet while also increasing their speed and accuracy.

3.2.2 Faster R-CNN

Faster R-CNN is a related object tracking algorithm to R-CNN. This algorithm uses the Region Proposal Network (RPN), which is more cost-effective than R-CNN and fast R-CNN at sharing full-image convolutional features with the detection network. A Region Proposal Network is a completely convolutional network that predicts object bounds and objectness scores at each object's location and is trained end-to-end to generate high-quality region proposals, which Fast R-CNN then uses for object detection [9].

3.2.3 SSD

The Single Shot Detector (SSD) is a deep neural network-based system for detecting objects in images. Over a range of aspect ratios, the SSD solution discretizes the output space of bounding boxes into a series of default boxes. The system scales per function map position after discretization. The Single Shot Detector network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes [9].

3.2.4 MobileNets

MobileNets are built on a simplified architecture that builds light weight deep neural networks using depth-wise separable convolutions. MobileNet proposes two basic global hyper-parameters for effectively balancing latency and accuracy. These hyper-parameters allow the model creator to choose the appropriate model size for their application based on the problem constraints. MobileNet presents detailed experiments on resource and

accuracy trade-offs, as well as high results on ImageNet classification when compared to other common models.

Table 2 demonstrates some objects mean average percentage detection accuracy of Faster R-CNN and SSD.

Table 2. Faster R-CNN and SSD Accuracy Comparison [26]

Number	**Category**	**Faster R-CNN**	**SSD**
1	airplane	84.9	85.6
2	bicycle	79.8	80.1
3	bird	74.3	70.5
4	boat	53.9	57.6
5	bottle	49.8	46.2
6	person	79.6	79.4

3.3 Existing Works on Person Detection and Tracking

Tracking and detecting of a person's location from a video surveillance have many applications such as security surveillance, tracking human behaviour and location in a specific location has attracted a great research attention. Over the decades, researchers are working in this field and improving the accuracy by applying different algorithms and methods. Detecting and tracking person have some challenges such as occlusion, changes of appearance, complex and dynamic background. Most of the early work relied on using conventional video camera with a lack of depth information.

Researchers used a single RGB-D camera for tracking and detecting person in real time [14]. The proposed method is three stage structure. Transforming the original RGB-D pixels to a novel Point Ensemble Image (PEI) from plan-view perspective to facilitate the subsequent two stages. In the second stage, the unsupervised detectors retrieve the positions that are human body plausible very quickly which are further refined by a classifier by using two new features Histogram of Height Difference (HOHD) and the joint Histogram of Colour and Height (JHCH). Finally, data association is carried out to the detection responses to generate the 3D trajectories. Experimental results show that this cascade structures offers very fast detection and tracking and produce a significant

performance advantage over existing methods. The proposed human detection and tracking system was implemented in C++ and the hardware used was Intel quad core i5 processor with 8Gb of ram.

Cognitive science approach is another method for detecting and tracking the human [15]. Because of the different variety of poses and appearance of human structure, doing human detection is quite challenging. This method is also efficient even if the background is cluttered. RGB camera is used for these purposes during night-time as there is limited amount of light and images are not clearly visible. Changing in lighting conditions is an important point as well. Already different methods for extractions of features have been developed for applying to SVM for classification. Histogram of Oriented Gradients which is mostly used for detecting human beings. Visual Saliency is also used [15]. Purpose of Visual Saliency is used for region proposal. Deep Multi-Level Network is also used because it has a encoder-decoder architecture which computes saliency maps as outputs. To train Support Vector Machine classifier, Histogram Oriented Gradient (HOG) features was used for detecting humans is any frames. For comparison, K-means algorithm was used and with the help of cluster HOG features from the consecutive frames set points in the image with common particular person moving around in the video can be identified. They achieved detection precision of 83.11% and a recall of 41.27%.

There are some image processing techniques for object tracking [16]. Tracking objects can be complex due to complex shape, motion and non-rigid nature. There are several tools available for tracking objects such as Blob tracking, Kernel based tracking, feature matching, Kalman filter, the Condensation algorithm, the dynamic Bayesian Network, the geodesic method etc. This tracking procedure is divided into four types: region-based tracking, active-contour-based tracking, feature-based tracking, and model-based tracking.

Fully Convolutional Region Proposal Networks is another approach for multispectral person detection [17]. Cameras are not restricted to the visual optical (VIS) spectrum. In the absence of light infrared cameras can reach higher detection performance compared to the VIS. Aim is to detecting persons in multispectral video that consists of three VIS channel (RGB) and one thermal IR channel. A novel fusion Region proposal network (RPN) which is built up on the pretrained very deep convolutional neural network VGG-16 [25]. The suggestion for using this network is assessed further by using a Booster

decision tree classifier to minimize the rate of false alarm detection. Improvement was by 18% as average miss rate was 29.83% on the test set of the KAIST Multispectral Pedestrian Detection Benchmark [24].

Another approach for real time person detection is with low resolution thermal inferred camera with Maximally Stable Extremal Regions (MSER) and CNNs [18]. MSER algorithm act as a hotspot detector when applied to Long-wave infrared Imagery. Because a person body temperature usually is higher than the background temperature. A drawback of MSER algorithm is, it cannot differentiate between different hot spots. Then by using CNN classifier the detected hotspots will be trained to discriminate between person hotspot and another hotspot. CNN designed for low resolution person hot spots which are found with LWIR imagery applications and it is also capable of instant classifications. LWIR person detection datasets depict an error rate which is reduced up to 80% by comparison with the last approach consisting of MSER [18].

For embedded platform OpenCV can be used for multiple object detection like human [19]. OpenCV is an open-source computer vision library that is used in real time computer vision. OpenCV is designed for real time applications. OpenCV covers hundreds of functions that cover robotics, image processing etc. Here presenting an overview of the cascade object detection algorithm as well as Haar-Like feature selection used by cascade classifier. Then proposed and OpenCV based solution for multiple object detection. The embedded system used in this work is the Texas Instrument DM3730 digital media processor.

CNN is also used for embedded platform for human presence detection by using low resolution thermal images [20]. Here presenting a new system that merge a low-resolution thermal camera with advanced feature extraction techniques like CNN. 32-bit micro controller has been used here. Grid eye device was used which was able to sense 8x8 thermal image every 10hz. A CNN was implemented in an ARM-cortex M4 core-based microcontroller by using the output of the sensor. The system was trained by using a custom dataset composed by short thermal videos. Then CMSIS-NN library was exploited for neural network which was developed by ARM in 2018. Due to its recent release, there are still few works using this framework, which can achieve 4.6X improvement in runtime/throughput and 4.9X improvement in energy efficiency with respect to a basic C implementation [20]. This is enabled by the optimized DSP (Digital

Signal Processing) instructions, such as SIMD and MAC, integrated in M4 and M7 cores to perform workloads of the neural network [20]. Some of these instructions have been used to accelerate low-precision computation in neural networks. The results shows that 76.7% accuracy in the microcontroller requiring only 16.5mW in continuous classification mode and using 6kB of RAM [20].

Deep learning has gained a tremendous influence on how the world is adapting to Artificial Intelligence since past few years [22]. Some popular object detection algorithms are Region-based Convolutional Neural Networks (RCNN), Faster- RCNN, Single Shot Detector (SSD) and You Only Look Once (YOLO). Amongst these, Faster- RCNN and SSD have better accuracy, while YOLO performs better when preference is speed over accuracy [22]. Deep learning combines SSD and MobileNets to perform efficient implementation of detection and tracking. This algorithm performs efficient object detection by not compromising on the performance [11]. In artificial vision, the neural convolution networks are recognized in the classification of images. Here researchers implemented an SSD and MobileNets based algorithms for detecting and tracking in python. Object detection involves detecting region of interest of object from given class of image [22]. Different methods are –Frame differencing, Optical flow, Background subtraction [22]. This is a method of detecting and locating a moving object by an camera.

Camera-based surveillance systems has inspired many researchers to research on human localization. Here, proposed a device-free localization method by using panoramic camera and indoor map [21]. The proposed method can provide precise location information of human object without any terminal device. Then searched all the foreground pixels to find whose location can represents the object's location best. The pixel location on the image then mapped to associated location on an indoor map. Finally, localization coordinates are obtained in the coordinate system we construct. Experimental results show that our localization method does not need any terminal device and is able to achieve a mean error of 0.37m, which is far less than those of popular fingerprinting and trilateration localization methods. With the development of mobile computing and popularity of smart terminal devices, location-based services (LBS) have played an important role in common people's life because it is able to offer accurate localization information for the services, such as object localization, emergency rescue and social network. Owing to signal attenuation and complex radio propagation, the performances of satellite and cellular based localization systems are very limited impacts in indoor

environments [21]. So various indoor localization systems have been developed for offering indoor LBS using the techniques like ultra-wideband (UWB), ultrasonic, radio frequency identification (RFID), wireless local area network (WLAN), and vision processing. Currently, many camera-based localization methods have been developed. Camera based localization and trajectory reconstruction were formulated as an optimization problem and this problem was solved by singular value decomposition (SVD) [21]. L. Liu et al. investigated the localization-oriented coverage problem of camera network and formulated the localization problem based on Bayesian estimation to obtain the needed camera density. Through incorporating camera parameters and some known location information, proposed a location-constrained maximum a posteriori algorithm for camera-based localization. Then Y. S. Lin et al. proposed a series of image transforms based on the vanishing point of vertical lines [9]. The transforms improved the proposed probabilistic occupancy map (POM)-based human localization in both outdoor and indoor environments.

3.4 Drawbacks of Existing Works

Purpose of this thesis is to detect the person location and tracking by camera with the computational power of embedded hardware. Very few researchers had worked on object detection algorithms which was purposed for embedded hardware. Most of the algorithms and models were tested on high-end configuration devices. Accuracy and speed were much faster on those devices. Most of the models only detect the objects and track the object. This does not satisfy our thesis purpose. There are not many models which can detect a person distance from the camera in the indoor condition. Not only distance but also tracking the person according to the location is not available in the existing works which is intended for working on embedded hardware.

4 Detection and Tracking

There are various ways for person detection and tracking. Person detection refers to many terms such as face detection, movement detection, location of the person detection and so on. This detection can be done within a specific location, place or in a fixed camera frame.

Nowadays, many CNN has already developed which is specifically designed for running efficiently on embedded hardware. These CNNs are already trained with a large data sets and able to detect many objects including human. OpenCV is also used for object detection. The output data from this operation can be used for defining the location of the object from the camera. Location refers the distance from the camera which represent as X, Y coordinates. These coordinates can be further used to track the person inside the camera frame by drawing a line in the image.

4.1 Person Detection

Person detection is the initial step for this application. First, determining the existence of the person in the camera frame is the main task. Detecting person is the similar way of object detection. Here in this project, MobilenetV3-SSD CNN architecture is used. This CNN has already been trained with COCO datasets. Version 3 of MobileNet is much faster than previous versions. It is faster and provides high accuracy. A trained and config file is required. OpenCV allows to capture image or video feeds. OpenCV has many libraries. Here Deep Neural Network (DNN) module of OpenCV is being used along with a DetectionModel class. DetectionModel allows to set params for pre-processing input image. [11]. DetectionModel creates net from file with trained weights and config, sets pre-processing input, runs forward pass and return result detections. [11]. SSD topology is supported by DetectionModel. The detection result is shown by drawing a rectangle around the person. Figure 6 demonstrates how COCO dataset, OpenCV and MobileNet is used during the implementation of person detection.

```
thres = 0.55  # Threshold to detect object
#
cap = cv2.VideoCapture('test_video.mp4')
fps_start_time = datetime.datetime.now()
fps = 0
total_frames = 0

cap.set(3, 1280)
cap.set(4, 720)
cap.set(10, 70)

# img = cv2.imread('ehsan2_20.jpg')

classNames = ['person']
classFile = 'coco.names'
with open(classFile, 'rt') as f:
    classNames = f.read().rstrip('\n').split('\n')

configPath = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
weightsPath = 'frozen_inference_graph.pb'

net = cv2.dnn_DetectionModel(weightsPath, configPath)
net.setInputSize(320, 320)
net.setInputScale(1.0 / 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
```

Figure 6. Person detection with opencv and mobilenet

4.2 Determining Location

Determining location refers to many things such as the presence of the person in a room or the distance from the camera in a room within the camera frame. As GPS is not a suitable way to determine the person in indoor situation, our goal is to locate the person in a room with a camera and determine the distance of that person from the camera. Distance is represented as (X, Y) coordinates and it is updated in real time according to the movement of the person.

4.2.1 Defining Location

The location is needed to be defined and it is uncertain that the indoor or the space. So, the only known thing is that there is a camera from where the video feeds are taken. The approach here is to define the floor of the room or the space as 2D space. As the camera is the key and only known component of any environment, we define it as the centre of that 2D space. If we think of the whole plane as a 2D graph where we can determine any location by (X, Y) value and the camera is in the centre which means the coordinate of the camera is (0,0) then the representation is easily generalized for most of the situations.

In the figure 7, we visualize how we define location from the perspective of the camera. Here, we assume the X axis as when the camera angle will be calculated because the angles should be given in a config file. The axis X will be assumed by considering the wall or plane on which it will be attached with. So, the Y axis will be calculated by a perpendicular axis as the camera's X value will be 0 on that axis. This representation will work for various shapes of rooms as we determine the axis comparing the position and focus of the camera. The units will be the same as the ones we use for measurements: meter, centimetre, or millimetre. The calculation should not be dependent on specific measurement units, as it may be used in different scenarios.

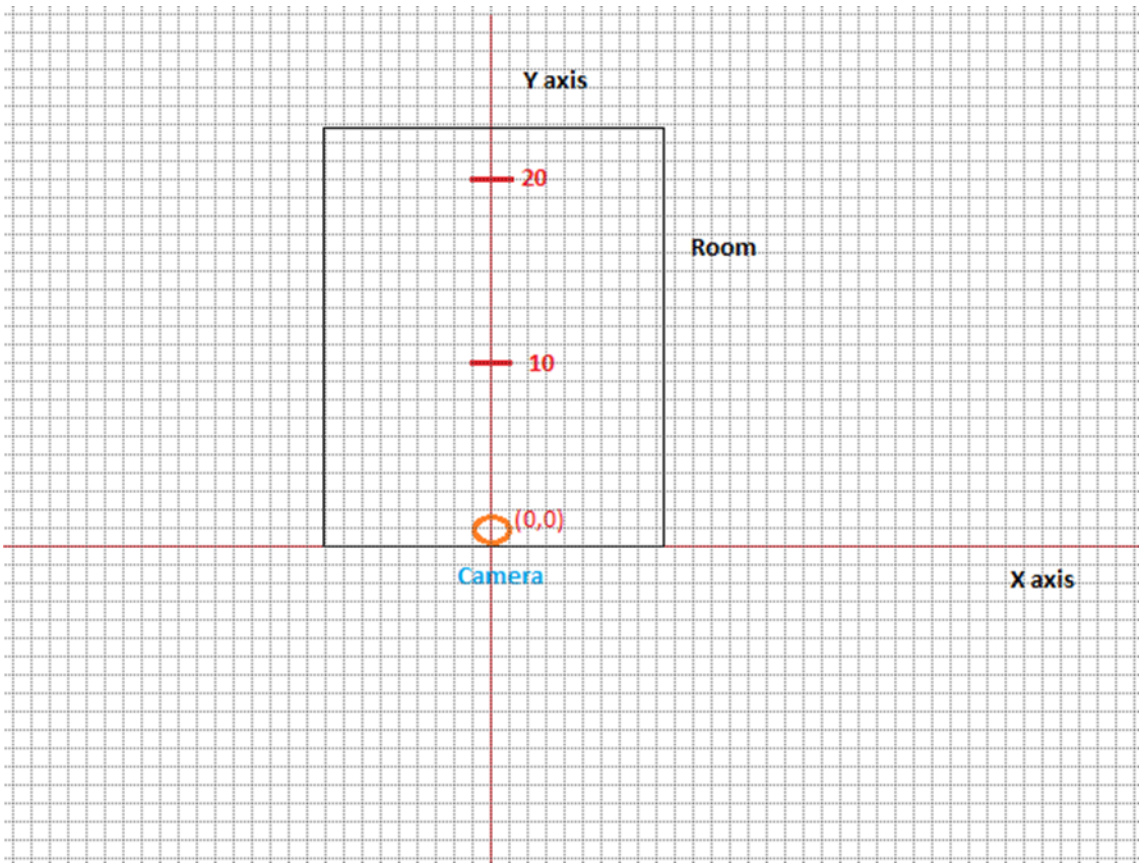

Figure 7. 2D plain of the indoor location where the camera is placed.

4.2.2 Approach of Location Calculation

From the video feed, we previously explained how we detected persons and what kind of information we have from that. In each frame, the detection method gives us the position of the pixels, where each person is located at. It provides: (x,y) the pixel location and height and width which represents a rectangular shaped area where a person is located in the frame. From the rectangle area it is easy to assume that the feet of a person will be on the bottom and if we calculate distance from the camera to the specific person, it will be measured from the feet to the bottom of the wall where the camera is attached. So if we can determine an ideal pixel which should be targeted for the calculation of distance or location it gives us an interesting approach.

We showed in figure 7 to explain a 2D plane by assuming the camera on the (0,0) position on the plane. All the cameras have the view angles, and it does cover a certain area. We assume the same x axis here and calculate view angles from left to right of the camera

means in the left side of the camera it is 0 degree and when it goes more righter, it increases. The figure 8 is a visual representation of the horizontal angles of the view angles of the camera.

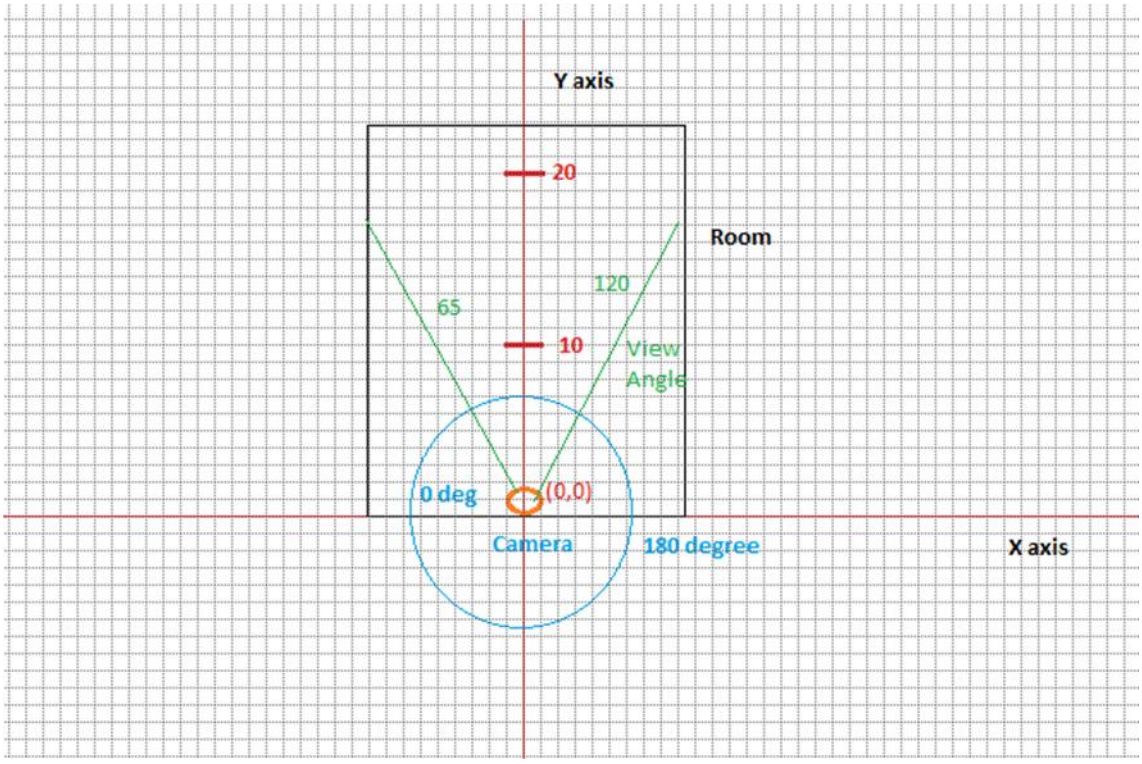

Figure 8. Horizontal angle representation of the room in 2D plain where camera is in centre.

So, as what we discussed before, the left of each frame starts from the left angle of the horizontal view and goes to the right angle of the horizontal view. Figure 9 represents the vertical angles of the view angles of the camera.

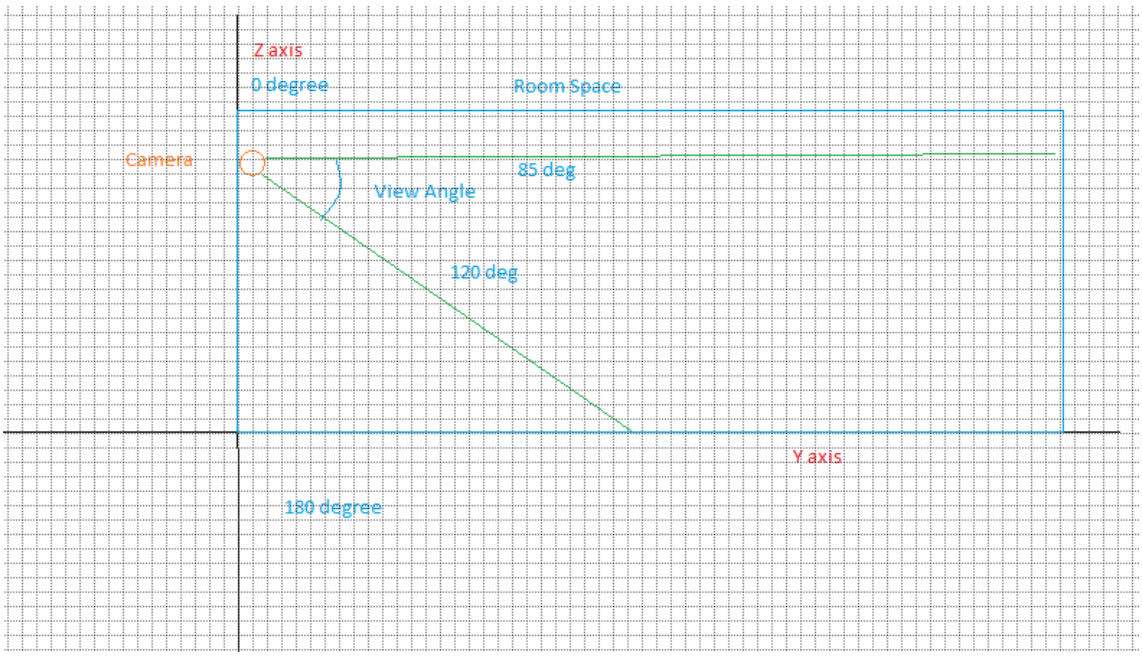

Figure 9. Vertical angle representation of the room assuming the Z axis on the 2D plain.

As we live in a 3D world(at least how it is visible to us) and the camera takes a frame which has 3D representation. Early we explained horizontal view angles of the camera by assuming the floor 2D and the camera focusing at the room in that 2D plane but in reality, the camera has more view angles: vertical view angles and diagonal view angles. For this purpose, we need the vertical view angles. In the figure given above the camera is in the Z axis where the value of z is not 0 because the plane of the floor is assumed as 0. From this perspective the camera has different angles as well. We assume 0 degree on the Z axis upper of the camera, when it comes down clockwise the value of the degree increases.

In the above explanation we defined vertical and horizontal view angles that a camera has and we defined a 2D plane where the camera is on (0,0) and has a value of Z which determines the height from the floor to the camera. For our calculation purpose these 5 values : left to right view angles (horizontal) and top to bottom view angles (vertical) and the height of the camera from the floor must be given as input. Our system takes a config file as input where the values can be written for any situation. To get the appropriate result from the system the from input should be given.

Now in the calculation approach we used, we need to measure the angles we discussed above and need to know the camera's height from the floor. As we discussed above, when we get the person's detected data as a rectangular part of the image, we simply take a

pixel which is in the middle of the bottom of the rectangular area. So, the (pixelX, pixelY) will be.

$$pixelX = x + (width / 2) \quad (4.2.1)$$

$$pixelY = y + height \quad (4.2.2)$$

In the equation 4.2.1 and 4.2.2, (x, y) are the upper left of the rectangular shape and width and height are given.

Based on this calculated pixel, at first, we calculated the Y value of the person detected. So, for the vertical angles we can say that the top index will start from the upper angle and the bottom will cover up to the bottom angle. So, for any pixel coordinate, if we know pixelY we can know the vertical angle of the camera for that pixel. We can do it by using a simple unitary method. Here is the calculation:

$$VerticleAngle = (UpperAngle - LowerAngle) / ImageHeight * PixelY + LowerAngle \quad (4.2.3)$$

From the camera this angle creates a right triangle and as we know the height of the camera so the distance of the person can be easily calculated with a simple trigonometry calculation.

$$distance = \tan\theta * base \quad (4.2.4)$$

Where, θ means VerticalAngle which is calculated in equation 4.2.3.

base is the camera height from the ground which is mentioned in equation 4.2.4. Figure 10 is given to clarify the equations 4.2.3 and 4.2.4.

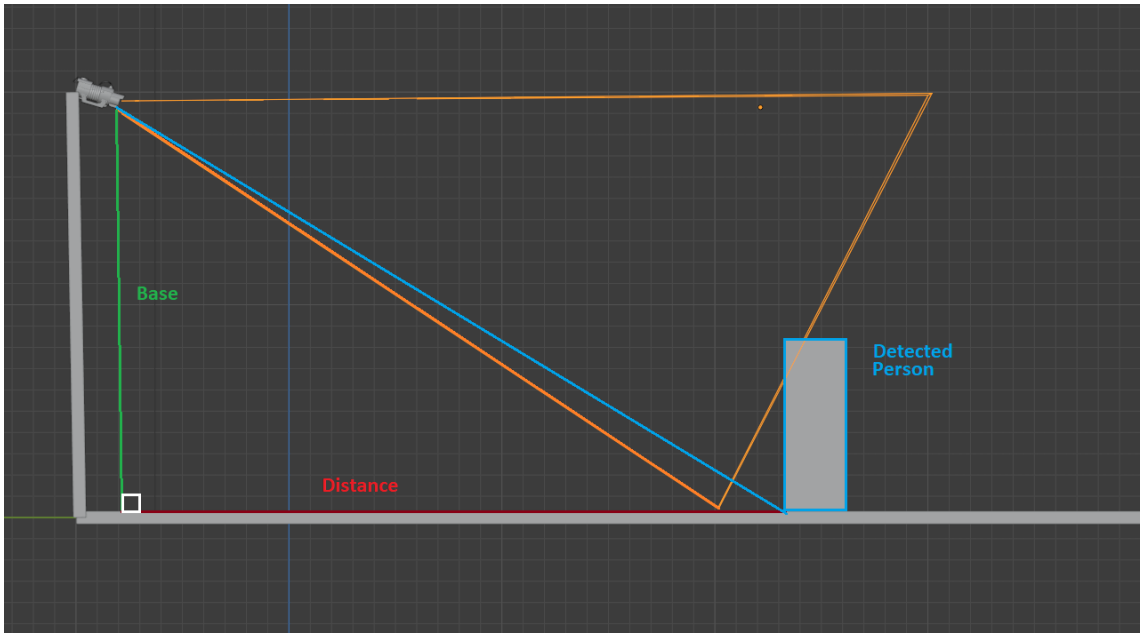

Figure 10. Y or distance calculation from the camera

The distance is the value of Y in the 2D plane where we are calculating. Now as we know the Y value or distance, we can use similar methods to calculate the X value as well. For calculating horizontal angle, we used unitary method as similar as before,

$$HorizontalAngle = (RightAngle - LeftAngle) / ImageWIdth * PixelX + LeftAngle \quad (4.2.5)$$

And for calculating X we can use the same tan function as well.

$$X = tan\,\theta * distance \quad (4.2.6)$$

In the equation 4.2.6, distance is equals to Y. and theta will vary if HorizontalAngle's value which is calculated in equation 4.2.5 is greater or less than 90 degree,

If HorizontalAngle > 90 then

$$\theta = HorizontalAngle - 90 \quad (4.2.7)$$

Else,

$$\theta = 90 - HorizontalAngle \quad (4.2.8)$$

Figure 11 represents the visualization of X distance that we calculated based on Y distance.

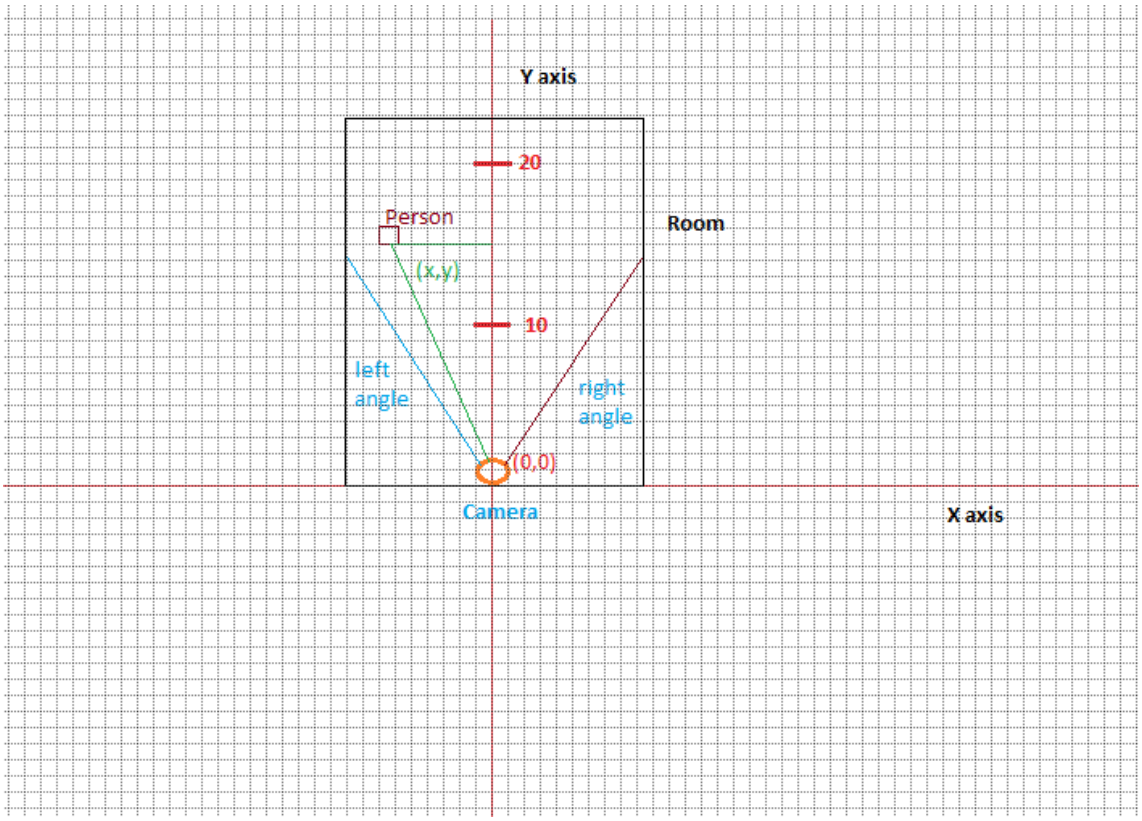

Figure 11. X Distance Calculation

By this way, we calculated (x, y) coordinates of the location of the detected person.

4.2.3 Implementation of Location Calculation

The calculation of distance between the detected person and the camera depends on only four simple methods. The descriptions are given below:

Function parameters takes in the position of a given pixel, the length of the entire image/frame, the lowest possible angle respective to the camera and the highest possible angle respective to the camera from the visible frame in any direction. The purpose Figure 12 is to find out the angle of a given pixel with respect to the camera and the ground .

```
def calculateAngle(pixel_length, image_length, low_angle, high_angle):
    netvalue = high_angle - low_angle
    multiplyer = netvalue / image_length
    result = multiplyer * pixel_length
    return result + low_angle
```

Figure 12. Angle Calculation

The purpose of Figure 13 method is to calculate the distance between the person and the camera on the ground.

```
def calculate_height_right_triangel_tan(base, angle):
    angle_radian = degree_to_radian(angle)
    return math.tan(angle_radian) * base
```

Figure 13. Distance Calculation Between Person and Camera

Figure 14 method utilizes the previously mentioned three methods to calculate the distance between the plane below the camera and the position of the person. The two angles that are being calculated are the vertical and horizontal view portion of the camera with respect to its own 180-degree angles in each direction to calculate the distance from X value and Y value, respectively.

```
def calculateDistance(upper_angle, lower_angle, left_angle,
right_angle, image_height, image_width, camera_height, box):
    pixelX = box[0] + (box[2] / 2)
    pixelY = box[1] + box[3]
    vertical_angle = calculateAngle(pixelY, image_height, upper_angle,
lower_angle)
    horizontal_angle = calculateAngle(pixelX, image_width, left_angle,
right_angle)
    distanceY = calculate_height_right_triangel_tan(camera_height, 180
- vertical_angle)
    if (horizontal_angle < 90.0):
        distanceX = calculate_height_right_triangel_tan(distanceY, 90
- horizontal_angle)
        distanceX = -distanceX
    elif (horizontal_angle > 90.0):
        distanceX = calculate_height_right_triangel_tan(distanceY,
horizontal_angle - 90)
    else:
        distanceX = 0.0
    return [distanceX, distanceY]
```

Figure 14. Calculate Distance

4.3 Tracking Person Based on Detected Location

4.3.1 Finding Individuals in Consecutive Frames

To track a person, we need to identify the same person from the previous ones. So, as we detect a new person, we tag the person with a unique ID. and in the next frame it returns the same ID if the person occurs. We used a non max suppression method and an object tracking algorithm called centroid tracking to track the person with a unique ID. Non-Maximum Suppression is a computer vision method that selects a single entity out of many overlapping entities (for example bounding boxes in object detection). The criteria are usually discarding entities that are below a given probability bound [13]. This object tracking algorithm is called centroid tracking as it relies on the Euclidean distance between (1) existing object centroids (i.e., objects the centroid tracker has already seen before) and (2) new object centroids between subsequent frames in a video [12]. As we tag the individual with an ID, then we can track the location changing in each frame or after several frames and add the latest location in its data set.

4.3.2 Tracking Mechanism

As in the previous section we explain how we detect the same person in consecutive frames, the next challenge was to store and represent the data in a meaningful way. As we get new locations per frame, we tried to implement some data structures to maintain the data. Figure 15 demonstrates the class that we created.

```
class Person:
    def __init__(self,objectID = 0):
       self.objectID = objectID


    def test(self):
        print(self.objectID)

class Location:
    def __init__(self,box =[0.0, 0.0, 0.0,
0.0],result=[0.0,0.0],timeStamp=''):
        self.box = box
        self.result = result
        self.timeStamp = timeStamp

    def testLocation(self):
        print(self.box,self.result)

class Track:
    def __init__(self, person = Person(), status=True, clear_counter
=0, save_counter = 0, draw_counter = 0):
        self.person = person
        self.status = status
        self.clear_counter = clear_counter
        self.save_counter = save_counter
        self.draw_counter = draw_counter
        self.list =[]

    def addValue(self, location = Location()):
        self.list.append(location)

    def test(self):
        for x in self.list:
            print(self.list.index(x))
            x.testLocation()
```

Figure 15. Person, Location, Track Classes

The Track object refers to a specific person's location at a different time. After a specific number of frames, we add a new location to the track object. And draw it in the view to show the location changing over the time. We maintain another array of Track object to track multiple objects. If the video feed is running in N fps, then it will save N location objects for that person each second, but which may cause data overloading because we are saving the data in the array list. So, we tried to take data after each 15 frames or 75 to reduce data and which would be meaningful while drawing. Also, it will cause less drawing on the screens. When a person goes out of the frame. We clear the data of his previous location and dump it into the Comma separated value (CSV) file as it works as a log file.

Figure 16 is used for update person location and tracking.

```
def updatePersonAndLocationAndDraw(img,person,location):
    for p in trackPersons:
      if (p.person.objectID == person.objectID):

          p.status = True

          p.save_counter = (p.save_counter + 1) % data_saving_fps
          if ((p.save_counter) ==0):
            p.addValue(location)

          return

    track = Track(person)
    track.save_counter = 0
    track.addValue(location)
    trackPersons.append(track)
```

Figure 16. Location Update and Tracking

Figure 17 method is used for drawing the tracking line in the image.

```
def drawAllTracks(image):
    for p in trackPersons:
        drawTrackPerson(image, p.list)
```

Figure 17. Draw Track Line

Figure 18 method is used for check if the person is present in the frame. If the person left out of the frame, location coordinates of that person get writes into the CSV and remove from the array list.

```
def clearUnusedPerson():

    removeList = []

    for q in trackPersons:
        if (q.status == False):
            q.clear_counter = q.clear_counter+1
        else:
            q.clear_counter = 0

        if(q.clear_counter == 5):
            removeList.append(q)

        q.status = False
        # print(q.person.objectID)

    for r in removeList:
        writeToCsv(r)
        trackPersons.remove(r)
```

Figure 18. Person status check and Log Writing

4.3.3 Log for Tracking Data

In the previous section 4.3.2, we mentioned that we maintain location data for each person while the person is on the frame. But when the person goes out of that frame, we do not need that person's data to be stored in the RAM. Moreover, we want to store and represent the data we are working with for later work. It is possible that we store that data in the storage and clear the memory that it can be used for further computation. Also, the

dumped data should be representable or should maintain a structure that we can use it to represent something meaningful later. Our solution is to dump the data in a csv file. The figure 19 is showing a part of a sample csv file our system is generating.

Person ID	Timestamp: Location(X,Y)	Timestamp: Location(X,Y)	Timestamp: Location(X,Y)
ID #0	2021-04-07 12:22:16:(-14.121955,255.681012)	2021-04-07 12:22:17:(-11.063921,255.681012)	2021-04-07 12:22:17:(-5.455050,250.418886)
ID #1	2021-04-07 12:21:09:(-2.740847,322.031640)	2021-04-07 12:21:10:(-1.682730,227.519969)	2021-04-07 12:21:11:(-2.463216,224.653709)
ID #2	2021-04-07 12:22:34:(13.309120,264.911819)	2021-04-07 12:22:35:(6.143192,262.079483)	2021-04-07 12:22:36:(-2.606292,270.747725)
ID #3	2021-04-07 12:22:37:(12.997405,253.904841)	2021-04-07 12:22:38:(6.566845,257.479896)	2021-04-07 12:22:39:(0.250798,259.301947)

Figure 19. Sample csv file

The format we chose to write on that file is for each person we add a new row to the file. Each row contains the whole data for the person before the person leaving the camera frame. So, the number of rows depends on the number of persons appearing in the frame and left. The number of columns that each row contains depends on the amount of time the person appears on the frame. That means if person A stays 15 seconds longer than person B then the row of person A will contain 15(approx.) more columns than person B's data in the csv file, as we are saving location after 1 second or 15 frames for each person. Image in figure 20 will show a row of what data what column will represent and the next figure 21 will show how a row of real data looks like in the csv.

Person ID	Timestamp: Location(X,Y)	Timestamp: Location(X,Y)	Timestamp: Location(X,Y)

Figure 20. CSV data for each row.

ID #0	2021-04-07 12:22:16:(-14.121955,255.681012)	2021-04-07 12:22:17:(-11.063921,255.681012)	2021-04-07 12:22:17:(-5.455050,250.418886)

Figure 21. Sample row represents the data of a person.

The method we wrote to dump the data to the csv file is given in figure 22. After a person left the frame, we delayed for 5 more frames before calling the method for dumping as it helps to make sure that the person is really gone from the video frame.

```
def writeToCsv(trackPerson):

    # print(trackPerson.person.objectID)

    with open('log.csv', 'a') as f:
        writter = csv.writer(f)
        list = []

        list.append("ID #"+str(trackPerson.person.objectID))
```

Figure 22. Data Save in CSV file

5 Experiments and Results

To begin with, we set up the threshold value for the person detection. Threshold value determines the detection percentage. For example, if a person is detected in the camera frame, accuracy of the person detection in set by the default. This threshold value plays a vital role in the experiment. To get the desire result in different light condition and environment, we change the value to get the maximum detection rate. In the Figure 23, the person was detected, and the accuracy of the detection is showing 61.9%. Here, we set the threshold value 0.55. That is why system is detecting the person with low accuracy.

Figure 23. Detected person shown by a green rectangle on the frame.

In the figure 24, threshold value was set to 0.80. So, the detection was started when percentage rate was above 80%. Though the person detection accuracy is much better, but the system detects the person when they are very close to the camera.

In a JSON file, we stored the value of our camera configurations and height. Configuration consists of upper angle, lower angle, left angle, and right angle. Height value from the ground where the camera is located also stored here. Angles were measured in degree and height can be measured in any unit. Unit of the height value determines the unit of the distance coordinates. For instance, in the figure 24, if the camera height is set to 81 centimetres, then the distance is shown X = 1cm and Y = 223cm.

Figure 24. Calculated location is shown of the detected person in white text.

The coordinates from the location detection stage were stored in an array list temporarily. From the stored coordinates, we drew the tracking line inside the image. The line is drawn until the person is available inside the video frame. Once the person is out of the frame, tracking line disappears and the model starts tracking a new person again.

In figure 25, a person is detected in a scene. Detection accuracy is showing 61.55%. The person was moving in a direction. From the far left of the image, it is seen that a red line is following the person where the person was moving. The red line was started from the position where the person got detected initially in the frame.

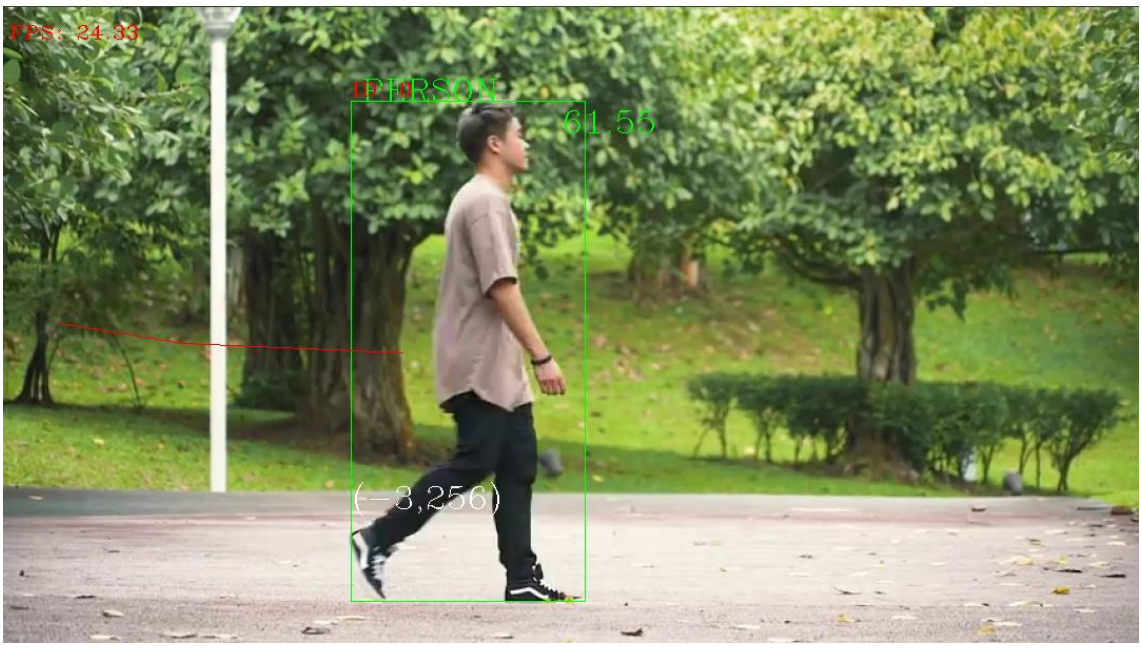

Figure 25. Real time location tracking by a red line

The model detects the person and tag the person with a unique ID. The ID count will rise ascending order until the system gets rebooted. First detected person in the frame is tagged as ID 0. From figure 23, figure 24 and figure 25 only one person is in the frame and it is tagged as ID 0. Figure 26 demonstrates the situation where two persons is detected as ID 1 and ID 2.

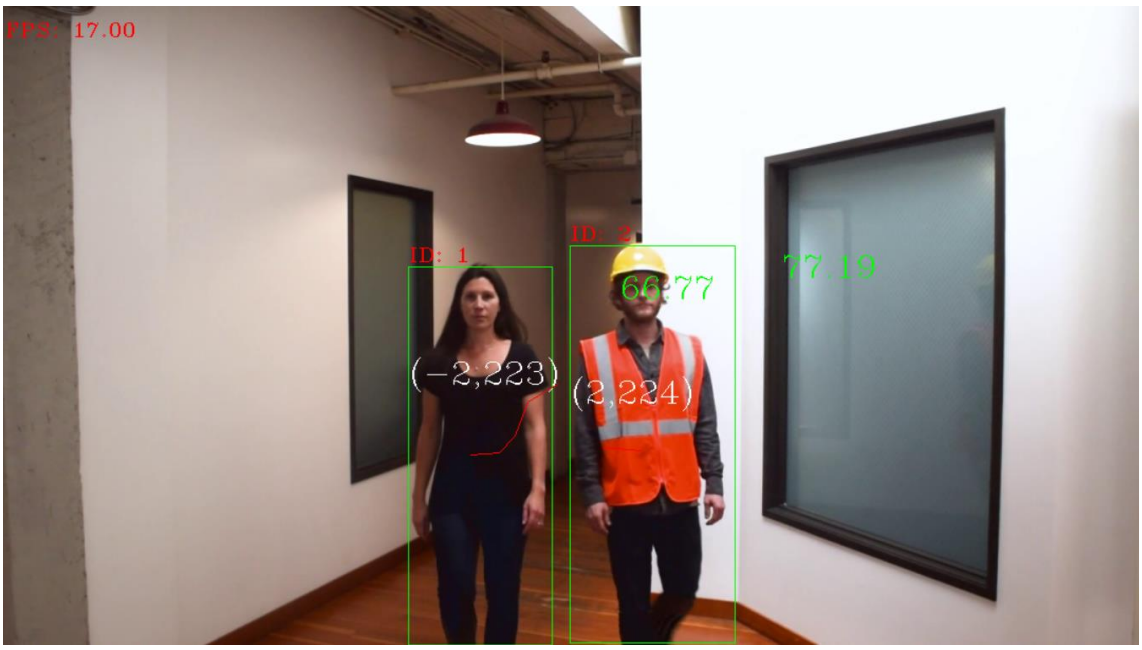

Figure 26. Detected persons tagged with unique ids.

This model provides around 15 fps to 25 fps from a webcam or a dedicated camera in real time. To make it more efficient, we did the calculation with a every 15 frames interval. So instead of detecting and tracking the person in each frame, we tracked a person in every 15 frames and stored the associated coordinates in the log file. Because of this approach, less data was stored in the memory and fps increased up to 5fps.

5.1 Limitation and Improvements

In this portion we talked about the limitations of this project and possible solutions/improvements to them. Working with a minimal hardware resource has some certain drawbacks that we simply could not overcome in such a short time. Some of them are related to the device itself and some of them are sheer natural.

5.1.1 Data Size

While processing video feed, the software is working with 15 fps video. For a single frame, we have a location object with 4 attributes. 4 byte per variable, total data size stands out to be 16 bytes. So, for each second our total data size will be 16 * 15 bytes or, 240 bytes. With this rate, in 100 seconds our data size will be 240 * 100 bytes or 24 kb.

This is a relatively large amount of data when we consider a room full of people in front of the camera and we will have to track every individual. As the RAM is limited, it will be an unoptimized mechanism for such scenarios.

To illustrate an example, if we keep on tracking one single person for 100 minutes, the size of the data will be 240*60*100 bytes or 1.44 mb which can cause an interruption when there are numerous individuals in front of the camera in places like hospitals or banks etc.

To optimize it, we tried to work with 5 fps video feed. In that case, the tracking of one single person for 100 minutes will take 480 kb which is marginally better than 15 fps video. The problem is, as we will be working on only 5 frames per second, a huge amount of data will be lost resulting in an inaccurate person tracking.

To get a better tracking output, we need to work with more fps which will cause us even more physical storage eventually making the system unusable within a very short time. In the current situation, our only way to get a reasonable result is to compromise frames.

5.1.2 Partial Body Detection

The calculation of the location of a person solely depends on specific pixels of the position of both the feet. If the upper half of the body of a person is detected but the lower part is not, it is not possible to specifically detect the accurate distance between the person and the camera. If there is a desk in front of a person where the feet are not visible, the system will detect that person but will not be able to provide the actual results from the camera. Distance will be incorrect. Hence, partial body detection cannot give us any usable result. One of our suggested method for improving this limitation is if we can calculate the detected body percentage then assume of the whole-body height to calculate the location.

5.1.3 Detection and Flicking

In the case of multiple people in the video, the person detection algorithm often loses the track of any random person and then starts counting him/her as a new individual although he/she was detected a few frames ago. This was seen only in cases where there were multiple people on the video. This is the issue from the detection model and needs further fixes to remove the flickering.

5.1.4 Cloud Storage Option and Security

Currently, the system works offline without any connection to the internet. It is good in a sense that no third party can hack into the system to abuse the camera but the downside of it is that the track of each detected person is storage in local memory without any backups. In case of any kind of storage failure such as read-write malfunction/accidental damage etc the data will be lost forever and there is no way to get them back. The safer option is to store the backups on any cloud storage which will require internet access and it will also raise the question of how secure the system is from being hacked by any third party. In this project, we did not take any security measures that will prevent hackers from taking control of the embedded system to misuse the cameras.

5.1.5 Hardware Limitations

A video feed monitoring system needs to be kept on 24 hours a day. As this is an embedded hardware-oriented work, keeping the system on all time will cause an internal heating and because of it being an embedded system, it is prone to crashing anytime the heat is over a certain threshold. This will interrupt the surveillance and will need a cooldown time for the machine to be ready to work again.

5.1.6 Works Only on Plain Ground

The current system works properly only if the person and the bottom of the camera is placed on the same plane. Hence, if the camera is placed on a plane ground but the person is on a staircase where his/her legs are on a higher ground, or on a lower ground, we will fail to measure the distance.

6 Summary

The purpose of this thesis was to find out a solution to track person movement in real time in indoor environment. To be able to do that the system requires to detect the person and try to calculate the location in real time and represent the location data in a meaningful way thus anybody can understand.

To get the result, object detection was performed by using Mobilenetv3-SSD CNN with OpenCV library. By depending on the camera position, we assume a 2D plane where the camera will be in the centre of the room and we represented the location with X, Y coordinates. Then we invented an approach for calculating the location from detected person's data where viewing angles and height from the floor of the camera is known. Detected person is tagged with a unique ID and update that individual's location in real time and draw on the video to show tracking.

The result of the calculated data showed not more than 5 percent error in any frame we computed, where the comparison was made with the hand calculated distance with the distance calculated from data. The fps varies from 15 to 25 when using real time camera feed and camera resolution also affect the fps as we found for 640p resolution fps is 20-25 and for 1080p resolution fps is 15-18. The model works accurately when there are up to 4 persons in the frame at the same time.

For future improvements, one of the main areas is when a person partially appears in a frame which we say partial human detection. Because currently the system is providing inaccurate results when the partial body of a person is detected. Another improvement is to make the project work in every scenario because currently the system is providing accurate results only if a person is standing on a plain ground. Another improvement that we should work on is to use a better detection algorithm or use the current algorithm by fixing bugs like sometimes it fails to detect a person in some frames which affects the tracking data that we are saving.

7 References

[1] [Online]. Available: https://www.mygreatlearning.com/blog/object-detection-using-tensorflow/.

[2] [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2015.00029/full.

[3] [Online]. Available: http://www.ece.ucy.ac.cy/labs/easoc/People/kyrkou/Projects/M.Sc.%20Thesis.pdf.

[4] [Online]. Available: https://ebrary.net/22041/computer_science/typical_architecture_embedded_system.

[5] [Online]. Available: https://steelkiwi.com/blog/python-for-ai-and-machine-learning/.

[6] [Online]. Available: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-73003-5_35#:~:text=Human%20detection%20is%20the%20task,templates%20or%20patterns%20of%20people..

[7] [Online]. Available: https://qengineering.eu/deep-learning-with-raspberry-pi-and-alternatives.html.

[8] [Online]. Available: https://asp-eurasipjournals.springeropen.com/articles/10.1186/1687-6180-2013-176.

[9] [Online]. Available: https://analyticsindiamag.com/top-8-algorithms-for-object-detection/.

[10] [Online]. Available: https://www.infopulse.com/blog/challenges-and-issues-of-embedded-software-development/.

[11] [Online]. Available: https://docs.opencv.org/master/d3/df1/classcv_1_1dnn_1_1DetectionModel.html.

[12] [Online]. Available: https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/.

[13] [Online]. Available: https://paperswithcode.com/method/non-maximum-suppression#:~:text=Non%20Maximum%20Suppression%20is%20a,below%20a%20given%20probability%20bound..

[14] Y. L. G. Z. P. Z. Y. Q. C. Jun Liua, “Detecting and tracking people in real time with RGB-D camera.,” 2015.

[15] A. G. Y. K. Vandit Gajjar, “Human Detection and Tracking for Video Surveillance: A Cognitive Science Approach,” 2017.

[16] S. S. Shipra Ojha, “Image processing techniques for object tracking in video surveillance- A survey,” 2015.

[17] M. A. C. J. G. L. H. N. T. Daniel Konig, “Fully Convolutional Region Proposal Networks for Multispectral Person Detection,” 2017.

[18] T. M. D. W. J. B. Christian Herrmann, “Realtime person detection in low-resolution thermal infrared imagery with MSER and CNNs,” 2016.

[19] A. A. A. M. Souhail Guennouni, “Multiple Object Detection using OpenCV on an Embedded Platform,” 2014.

[20] R. P. E. F. Gianmarco Cerutti, “CONVOLUTIONAL NEURAL NETWORK ON EMBEDDED PLATFORM FOR PEOPLE PRESENCE DETECTION IN LOW RESOLUTION THERMAL IMAGES,” 2019.

[21] J. L. G. B. Yongliang Sun, “Device-Free Human Localization Using Panoramic Camera and Indoor Map,” 2016.

[22] A. J. H. J. M. Chandan G, “Real Time Object Detection and Tracking Using Deep Learning and OpenCV,” 2018.

[23] [Online]. Available: https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework .

[24] [Online]. Available: https://soonminhwang.github.io/rgbt-ped-detection/.

[25] [Online]. Available: https://towardsdatascience.com/region-proposal-network-a-detailed-view-1305c7875853.

[26] R. C. M. Z. Y. P. Y. Z. M. Y. Baohua Qiang, “Convolutional Neural Networks-Based Object Detection Algorithm by jointing Semantic Segmentation for Images,” China, 2020.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Abu Ishtiaque Md Fahim

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis Camera Based Person Tracking and Location Detection on Embedded Hardware, supervised by Mairo Leier, co-supervised by Uljana Reinsalu.
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

10.05.2021

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.